NPS ARCHIVE
2000.06
LOPEZ, B.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN INVESTIGATION AND ASSESSMENT OF LINUX IPCHAINS AND ITS VULNERABILITIES WITH RESPECT TO NETWORK SECURITY

by

Bryan S. Lopez

June 2000

Thesis Advisor: Raymond F. Bernstein Jr.

Approved for public release; distribution is unlimited.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2000	Master's Thesis

4. TITLE AND SUBTITLE
An Investigation and Assessment of Linux Ipchains and its Vulnerabilities with Respect To Network Security

5. FUNDING NUMBERS

6. AUTHOR(S)
Bryan S. Lopez

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this research paper are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This research thesis formulates a survey of network security and IPChains, the Linux firewall. It provides a detailed description of prominent network security procedures in use today. This paper falls directly in line with the goals of Executive Order 13010, the President's Critical Infrastructure Protection Plan, supports the goals of the National Security Agency's SIGINT Business Plan and the goals of both the Unified and Maritime Cryptologic Architecture. It will aid in the development of the problem solving efforts of the national cryptologic organization and be used to provide critical intelligence support to the Operational command and the national intelligence community.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Linux, Network Security, Ipchains, Unified Cryptologic Architecture	137
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18, 298-102

Approved for public release; distribution is unlimited.

AN INVESTIGATION AND ASSESSMENT OF LINUX IPCHAINS AND ITS VULNERABILITIES WITH RESPECT TO NETWORK SECURITY

Bryan S. Lopez
Lieutenant, United States Navy
B.A., University of New Mexico, 1990
M.S., Troy State University, 1992

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2000

NPS ARCHIVE
2000.06
LOPEZ, B.

Thesis
L82244
c.1

NAVAL

ABSTRACT

This research thesis formulates a survey of network security and IPChains, the Linux firewall. It provides a detailed description of prominent network security procedures in use today. This paper falls directly in line with the goals of Executive Order 13010, the Critical Infrastructure Protection Plan, supports the goals of the National Security Agency's SIGINT Business Plan and the goals of both the Unified and Maritime Cryptologic Architecture. It will aid in the development of problem solving efforts of the cryptologic organization and be used to provide critical intelligence support to the Operational command and the national intelligence community.

TABLE OF CONTENTS

I. LINUX 1

A. GOALS OF RESEARCH 1
B. WHY LINUX? 2
C. INTRODUCTION 5

II. FIREWALLS 9

A. OVERVIEW 10
B. DMZ 12

III. TYPES OF FIREWALLS 15

A. OSI MODEL 15
B. SCREENING ROUTER (PACKET FILTERS) 16
C. PROXY SERVER GATEWAYS 17
D. CIRCUIT-LEVEL GATEWAY 17
E. APPLICATION-LEVEL GATEWAY 18

IV. SECURITY CONSIDERATIONS 21

A. STATEFUL INSPECTION TECHNIQUES 21
B. SECURITY POLICIES 22
C. SERVICE PORTS: KEY ACCESS POINTS 24

V. PACKETS: IP NETWORK MESSAGES 27

A. INTERNET CONTROL MESSAGE PROTOCOL (ICMP) 28
B. USER DATAGRAM PROTOCOL (UDP) 33
C. TRANSMISSION CONTROL PROTOCOL (TCP) 34

VI. TYPES OF ATTACK 37

A. ICMP REDIRECTS AND BOMBS 37
B. DENIAL OF SERVICE 38
C. SMTP SESSION HIJACKING 39
D. EXPLOITING BUGS IN APPLICATION 39
E. BUGS IN OPERATING SYSTEMS 39
F. SYN ATTACK 40

VII. RESEARCH 43

A. APPROACH 43
B. IPCHAINS 44
C. A TYPICAL ATTACK 46
D. HACKING TOOLS 47
1. NMAP 47
2. RED BUTTON 53
3. SHADOW ADVANTIS ADMINISTRATOR 58
4. SUB 7 60

VIII. SYSTEM CONFIGURATION 63

A. HARDWARE 63
B. SOFTWARE 65
C. ROUTING 65
D. SECURITY ASSESSMENT WITHOUT FIREWALL 67
E. BUILDING THE FIREWALL 72
F. SECURITY ASSESSMENT WITH FIREWALL 82

IX. CONCLUSIONS 89

A. SUMMARY OF FINDINGS 89

APPENDIX A: IPCHAINS RULES AND ACTIONS 91

1. IPCHAINS ACTIONS 91
2. IPCHAINS PARAMETER TYPES 92
3. OPTION CHOICES 92
4. IPCHAINS RULE SPECIFICATIONS 93

APPENDIX B: NMAP 97

APPENDIX C: GLOSSARY 113

LIST OF REFERENCES 123

INITIAL DISTRIBUTION LIST 127

I. LINUX

A. GOALS OF RESEARCH

This research thesis was conducted as a means of investigating the network security characteristics associated with the Linux operating system. A close examination of IPChains, the Linux firewall, was undertaken in pursuit of determining its relative vulnerability vis-a-vis the other popular networking software, Microsoft's Windows NT. This research falls directly in line with the goals of Executive Order 13010, the Critical Infrastructure Protection Plan and supports the goals of the National Security Agency's (NSA's) SIGINT Business Plan and the goals of both the Unified and Maritime Cryptologic Architecture. It will aid in the development of the problem solving efforts of the national cryptologic organization and be used to provide critical intelligence support to the Operational command and the greater national intelligence community.

The Director of the National Security Agency (DIRNSA), LT General Michael Hayden, has made it his personal goal to guide the nation's largest intelligence organization through a transitional period from the era of legacy-based systems and ideas to tomorrow's era of information technology-based intelligence exploitation. With the release of the National Security Agency's SIGINT Business Plan in March of this year, the foundations for a national cryptologic strategy for the 21st century were laid out. First and foremost among the many goals of this strategy is a shift in psychology, products and

services to those of an information technology environment. This thesis is a critical, early step in the direction of fulfilling NSA's goals of expeditious transition.

B. WHY LINUX?

In the last few of years and, more importantly, over the last several months there has been explosion of interest in Linux as an advantageous alternative to Windows-based client/server operating systems. In fact, there is speculation that in the years to come, Linux will overtake the Microsoft share of operating systems in the global market. In spite of the February 2000 release of Windows 2000, a recent IDC study indicates that Linux is growing and will continue to grow at a much faster rate than other operating systems. (Loshin, 2000)

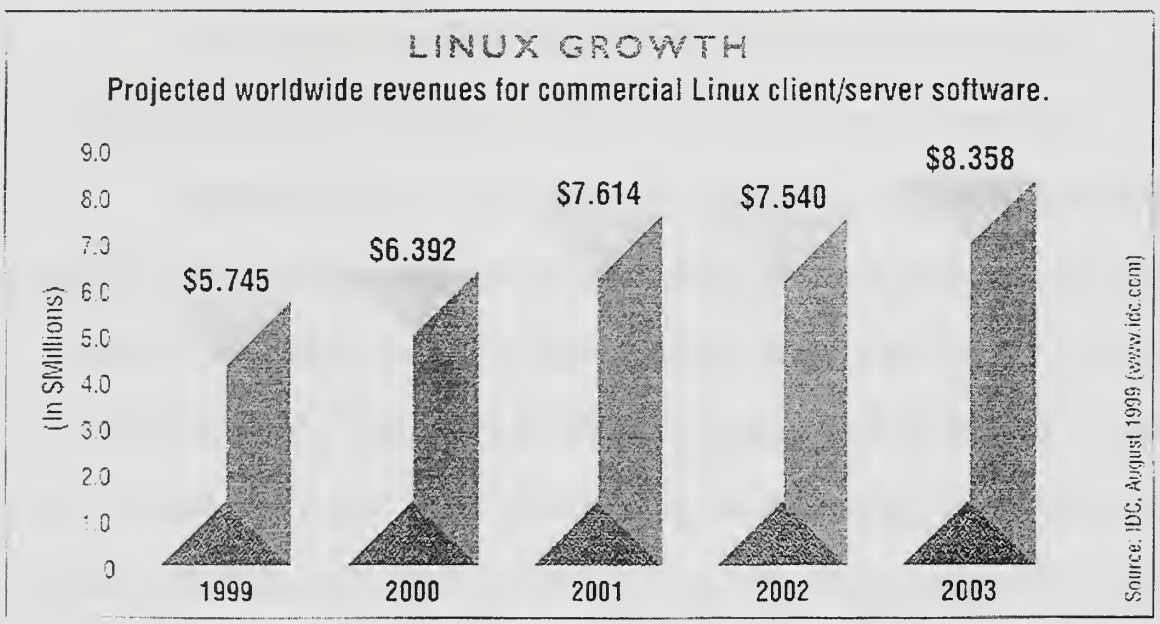

Figure 1. Linux Growth From Ref [Loshin, 2000]

If considering the influence of the sheer volume of sales potential an operating system could have in the world market, one could turn to China's recent commissioning of a Linux-based operating system as an example. Compaq China has recently begun the development of a simplified Chinese Edition of the Linux operating system at the request of the Chinese government. Compaq China will team up with Beijing Founder Electronics and the Institute of Software Academia Sinica's Sino-Software System Company to develop what is to be known as "Red Flag Linux." The impetus for the development of the system will be to support the People's Republic of China government and Chinese enterprises in their initiatives to go online.

The decision to commission such a versatile operating system and provide True Type simplified Chinese display and printing capabilities has some at Microsoft up in arms. There is concern within Chinese government and industrial circles that heavy reliance on Microsoft 2000 (and other Microsoft operating systems) could lead to security leaks and make government computers more vulnerable to viruses. Additionally, Sun Yufang, an official overseeing the Red Flag software project at the Chinese Academy of Sciences reported that government offices expressed strong interest in scrapping Windows, citing concerns Microsoft had the ability to access their computers over the Internet. Of course, Microsoft representatives were quick to issue a statement of rebuttal insisting that the Chinese assessments were unsubstantiated media reports. Nevertheless, it appears that China is, at the very minimum, considering a viable alternative to the use of Microsoft-based client/server operating systems. (Staff, 1999)

In addition to China's burgeoning interest in Linux-based operating systems, several other countries such as Germany, Israel, Taiwan, Japan and even Australia have begun to pursue research in the acquisition of Linux systems tailored to meet their specific needs. (Staff, 2000) (Staff Reporter, 1999)

From a Cryptologic perspective, adversarial interest alone might be impetus enough to conduct research into the intricacies associated with the myriad security issues linked to web-based Linux systems. However, more than looking from a strictly exploitation point of view, one must consider the security issues associated with a defensive posture. As the global community, and more specifically, the United States continues to rely on the Internet and networking technologies, there is an ever-increasing necessity to stay one step ahead of the competition in terms of network defensive measures.

Recent network global attacks perpetrated by small groups or even individual hackers have raised the concern of world governments from guarded interest to all out paranoia. While investigations are still underway, it appears that the perpetrators of the Love Bug virus that swept around the world with alarming speed and devastation were a couple of Filipino college students. The virus, which spread through twenty countries and caused more than ten billion dollars in damage, is the most devastating computer bug in history. (Maney, 2000) It is, at least, the second such action to affect global network activities in the past year.

A few months ago, so called "friendly" forces, those whose self-proclaimed purpose it is to bring security vulnerabilities to the public's attention, struck against some of the largest web sites in the world like, Ebay, Yahoo, CNN, Amazon, E*Trade and DATEK. Electronic commerce slowed to a proverbial halt as the distributed denial of service attacks took their toll on America's net-based cyber trading and information exchange sites. (Levy/Stone, 2000) Attacks like these should be heeded as alarms that represent the potential that exists for adversarial governments or terrorist organizations to damage network systems. Their actions have caused the governments of the world to take a much closer look at how they allocate their resources with respect to network defense. The continued exponential growth and reliance on network-based operations prove that network security is a problem that will never fade away. Network defense will only continue to be the Achilles' heel for American industry, military and government.

It is for this reason that this research is focused on the Linux operating system. The primary goal of this research was to provide background and investigate the security issues associated with the Linux operating system. For this task, I chose the latest release of a popular Linux software package, Red Hat Linux 6.1. In this thesis I will describe the installation and creation of a Linux-based network, provide some insights into Linux system administration and report the findings of research which will test known security vulnerabilities and kernel patches recommended for those vulnerabilities in the inherent Linux firewall code, Ipchains.

C. INTRODUCTION

The Linux operating system was actually the brainchild of Linus Torvalds, an undergraduate student from the University of Helsinki in Finland. (Sobell, 1997) It's an offshoot of the original UNIX operating system that was developed in the mid-1970's. Over the years, various organizations had tailored the UNIX code to meet the specific needs of their association. This was a very tedious process, not to mention the costs of commercial UNIX, which were phenomenally high for the time. Torvalds developed Linux as an alternative to the commercial code, something that would be accessible to the average user at a very inexpensive price, free! While it was originally developed as a hobby, something that programmers could tinker with, it quickly evolved into a powerful kernel that entire teams of software developers modified for the sheer joy of having a hand in its development. By 1992, it was officially released as a complete, fully operational UNIX-like operating system. (Danesh, 1999)

When using the term "Linux", caution must be used so as not to cause confusion. In one sense, Linux refers to the software "kernel", the heart of any version of Linux, but in another sense, it is often used to refer to the current distribution - the collection of applications than run on the kernel. For deconfliction purposes, in this thesis, Linux will be used to refer to the operating system in general. "Kernel" and "distribution" will be used for clarification purposes. (Ziegler, 2000)

In addition to being an inexpensive alternative to Windows-based operating systems, the characteristics that make Linux such an attractive option lie in its attractive

features such as multitasking and multi-user capability. Multitasking allows a computer with a single processor to appear to be performing multiple tasks simultaneously. While Windows systems like Windows 2000 and Windows NT are also multitasking, neither can compete with the robust multitasking capabilities Linux has to offer. Multi-user, as the word implies, allows multiple simultaneous users to log on to a network, thereby leveraging the multitasking capabilities of the operating system. This is a feature that has been inherent only to Unix operating systems until very recently with the release of Windows NT Terminal Server. (Shah, 2000)

Still, the average personal computer user and system administrator point to the ease of use of the graphical user interface (GUI) and the myriad of application software Windows systems have to offer. It turns out that Linux offers similar capabilities in their X Windows, built in scripting languages, word processing, databasing, DOS and Windows compatibility software all at a very reasonable price. The primary concern now is to investigate viable Linux-based security software for use by the average operator.

The principal concern when using any operating system in a network environment is its inherent security vulnerabilities. Linux is no more vulnerable than its Windows-based counterparts and, in fact, may offer added security measures not available in other systems. In addition to flat out, better built-in security prevention features, Linux also offers the advantage of "open source code", which provides the user with the option to find vulnerabilities and patch them at the root level. Windows-based systems require a

complete new software release from the manufacturer as Microsoft's code is not available to the root user. (Danesh, 1999)

When considering security issues from a networking perspective the primary point of defense from possible adversarial users or organizations is the network firewall. As previously cited, the Linux firewall, ipchains, will be the focus of this thesis, but bear in mind that there are many commercial firewall options available to both the Linux and Windows NT system administrator.

II. FIREWALLS

As many people know, a firewall is found in every car. In a car, the firewall is a physical barrier (usually metal) that separates the engine from the passengers. It is meant to protect the passengers in the car from any dangers should the engine catch fire, while allowing the driver continued access to the engine's controls. A firewall in a computer provides a similar service in that it protects a private network from possible "dangers" in the public domain of the Internet. Dangers can come in the form of ill-intentioned individuals or groups who would like to penetrate an internal network. Usually, one computer in an internal network will act as the "firewall," protecting the internal network from outside penetration. However, there are limitations with firewalls, primarily that the protected network can't reach the external Internet directly, but by the same token, external users cannot reach the protected network without first going through the firewall.

A firewall is an attempt to provide security. It aids in the implementation of a larger security policy that defines the permitted access and services. It implements security policy through network configuration, the use of host systems and routers, and other security measures such as advanced authentication in place of static passwords. A firewall system's main purpose is to control access to or from a protected network (i.e., a site). It performs network access security by forcing connections to pass through the firewall, where they can be examined and evaluated.

A firewall system can be a router or a host (i.e. a stand-alone personal computer), set up specifically to shield a site or local area network from protocols and services that

can be abused by hosts outside the local area network. A firewall system is usually located at a higher-level gateway, such as a site's connection to a wide area network (i.e., the Internet), however firewall systems can be at lower-level gateways and provide protection for some smaller collection of hosts or local area networks.

Firewalls can provide a measured level of confidence in network security, but are not the final answer in protecting a network. Firewalls cannot protect against attacks that emanate from within the network. Viruses are capable of penetrating firewalls. There are too many different methods of encoding binary files for transfer over networks and too many different viruses to protect from them all. Network users must be security conscious. Firewalls cannot prevent network users from allowing mail or copied files to be injected into and executed within the network causing data-driven attacks. Firewalls do an excellent job of keeping the outside world out of corporate networks but many research studies have shown that the threat from internal attackers is far more significant. (Cheswick, 1994)

A. OVERVIEW

A firewall puts up a barrier that controls the flow of traffic between networks. The safest firewall would block all traffic, but that defeats the purpose of making the connection, so the system administrator needs to strictly control selected traffic in a secure way. Firewalls are often described in terms of perimeter defense systems, with a so-called "choke point" through which all internal and external traffic is controlled. The usual metaphor is the medieval castle and its perimeter defense systems. The moats and

walls provide the perimeter defense, while the gatehouses and drawbridges provide "choke points" through which everyone must travel to enter or leave the castle. Access can monitored and blocked at these choke points.

The real threat is often the stealthy spy who slips over the castle wall in the dark of night and scales every barrier undetected to reach his target of attack. The primary question for any system administrator is how far do you let people into the internal network, and what do you allow them to do once inside? In keeping with our castle analogy, local townspeople and traders were usually allowed to enter the market yard of the castle with relative ease so they could deliver or pick up goods. At night, the gates were closed, and goods were brought into the castle--usually after close inspection. Following this correlation, the market yard could be compared to the public Web and FTP servers that you connect to the Internet for general availability. While just about anybody could enter the market yard, only trusted people and people with special credentials were allowed into the inner perimeters of the castle. Like the multiple perimeter defenses of the castles, multiple firewall devices can be installed to keep devious hackers out of your networks. "Trip wires" can be set up by installing "relatively weak" devices on the outer edge of your defense that sound alarms when attacked. Once in place, a firewall, just as a castle, requires constant vigilance. If someone can climb your fence at night when no one is looking, what good is the fence? Security policies and procedures must be put into place. In defending a castle, the archers and boiling oil men need a defensive strategy, and they need regular drills to ensure that the strategy works. If

your internal systems are hit by flaming arrows, you'll need a disaster recovery plan to put out the fire and get the system back online. Castle towers protect the soldiers who defend the castles. Without defenders, the castle is vulnerable to attackers that scale walls or knock down doors. Likewise, your firewall is not a stand-alone device. You need to manage and monitor it on a regular basis and to take action in the event of an attack. It is also only one part of your defense. If the attackers do get inside, you need to keep them from looting your systems by implementing security measures at each domain and server. This brings up another point. While firewalls are keeping Internet intruders out, your internal users might be looting your systems. You may need to separate departments, workgroups, divisions, or business partners using the same firewall technology, and you may need to implement encryption throughout your organization. Firewalls also do not protect against leaks, such as users connecting to the outside with a desktop modem. In addition, if some new threat comes along, your firewall might not be able to protect against it. Viruses and misuse of security devices are also a threat. (Cheswick, 1994)

B. DMZ

One answer to network security may be the use of a DMZ. "DMZ" is an abbreviation for "demilitarized zone". In the context of firewalls, this refers to a part of the network that is neither part of the internal network, nor directly part of the Internet. A DMZ can be created by putting access control lists on your access router. This minimizes the exposure of hosts on your external LAN by allowing only recognized and managed

services on those hosts to be accessible by hosts on the Internet. These services are not required for the operation of a web server, so blocking Transmission Control Protocol (TCP) connections on that host will reduce the exposure to a denial-of-service attack. In fact, if you block everything but Hypertext Transfer Protocol (HTTP) traffic to that host, an attacker will only have one service to attack.

A common approach for an attacker is to break into a host that's vulnerable to attack, and exploit trusts relationships between the vulnerable host and more interesting targets. If you are running a number of services that have different levels of security, you might want to consider breaking your DMZ into several "security zones". This can be done by having a number of different networks within the DMZ. For example, the access router could feed two ethernets.

On one of the ethernets, you might have hosts whose purpose is to service your organization's need for Internet connectivity. These will likely relay mail, news, and host DNS. On the other Ethernet could be your web server(s) and other hosts that provide services for the benefit of Internet users.

In many organizations, services for Internet users tend to be less carefully guarded and are more likely to be doing insecure things. This might be reasonable for your web server, but brings with it a certain set of risks that need to be managed. It is likely that these services are too risky for an organization to run them on a bastion host, where a slip-up can result in the complete failure of the security mechanisms.

By putting hosts with similar levels of risk on networks together in the DMZ, you can help minimize the effect of a break in at your site. If someone breaks into your web server by exploiting some bug in your web server, they won't be able to use it as a launching point to break into your private network if the web servers are on a separate LAN from the bastion hosts and you don't have any trust relationships between the web server and bastion host.

Keep in mind that we're running Ethernet here. If someone breaks into your web server, and your bastion host is on the same Ethernet, an attacker can install a sniffer on your web server, and watch the traffic to and from your bastion host. This might reveal things that can be used to break into the bastion host and gain access to the internal network.

Splitting services up not only by host, but also by network, and limiting the level of trust between hosts on those networks, you can greatly reduce the likelihood of a break in on one host being used to break into the other. In other words, breaking into the web server in this case won't make it any easier to break into the bastion host. In this way, routers become the first lines of defense in a comprehensive security strategy.

You can also increase the scalability of your architecture by placing hosts on different networks. The fewer machines that share the available bandwidth, the more bandwidth that each will get. (Cheswick, 1994)

III. TYPES OF FIREWALLS

A. OSI MODEL

In keeping with the interoperability goals as outlined by the Unified Cryptologic Architecture and mandated by NSA in May 2000, it is imperative that network security be given the highest priority. Since routers are considered to be a first line of defense for most networks, prioritization requires that a thorough understanding of how routers can be used as firewalls must be undertaken. Hence, a discussion of the various types of router-based firewalls ensues.

First, an introduction of the Open Systems Interconnection (OSI) network model is appropriate. The network model is comprised of seven layers, referred to as OSI. OSI provides for the transfer of information between end systems across some sort of communications network. It relieves higher layers of the need to know about the underlying data transmission and switching technologies used to connect systems. The lower three layers are concerned with attaching to and communicating with the network, while the others are concerned with the specific functions of data exchange.

OSI Model	
Layer 7:	Application Layer
Layer 6:	Presentation Layer
Layer 5:	Session Layer
Layer 4:	Transport Layer
Layer 3:	Network Layer
Layer 2:	Data Link Layer
Layer 1:	Physical Layer

Table 1. OSI Model [Stallings, 1997]

There are three major types of firewalls that use different strategies for protecting network resources. The most basic firewall devices are frequently built on routers and work in the lower layers of the network protocol stack of the OSI model. They provide packet filtering and are often called *screening routers*. High-end *proxy server gateways* operate at the upper levels of the protocol stack (i.e., all the way up to the application layer). They provide proxy services on external networks for internal clients and perform advanced monitoring and traffic control by looking at certain information inside packets. The third type of firewall uses *stateful inspection* techniques. (Siyan/Ware, 1995)

Routers are often used in conjunction with gateways to build a multi-tiered defense system, although many commercial firewall products may provide all the functionality necessary in one convenient package.

B. SCREENING ROUTER (PACKET FILTERS)

Screening routers can look at information related to the hard-wired address of a computer, its IP address (Network layer – layer 3), its service ports for TCP and UDP, header flags, and even the types of connections (Transport layer – layer 4) and then provide filtering based on that information. A screening router may be a stand-alone routing device or a computer that contains two network interface cards (dual-homed system). The router connects two networks and performs packet filtering to control traffic between the networks.

Administrators program the device with a set of rules that define how packet filtering is done. Ports can also be blocked; for example, you can block all applications

except HTTP (Web) services. However, the rules that you can define for routers may not be sufficient to protect your network resources, especially if the Internet is connected to one side of the router. Those rules may also be difficult to implement and error-prone, which could potentially open up holes in your defenses. (Siyan/Ware, 1995)

C. PROXY SERVER GATEWAYS

Gateways work at a higher level in the protocol stack to provide more opportunities for monitoring and controlling access between networks. A gateway is like a middle-man, relaying messages from internal clients to external services. The proxy service changes the IP address of the client packets to essentially hide the internal client to the Internet, then it acts as a proxy agent for the client on the Internet.

Using proxies reduces the threat from hackers who monitor network traffic to glean information about computers on internal networks. The proxy hides the addresses of all internal computers. Traditionally, using proxies has reduced performance and transparency of access to other networks. However, current firewall products solve some of these problems. There are two types of proxy servers, circuit-level gateways and application-level gateways. (Siyan/Ware, 1995)

D. CIRCUIT-LEVEL GATEWAY

The circuit-level gateway type of proxy server provides a controlled network connection between internal and external systems (i.e., there is no "air-gap"). A virtual "circuit" exists between the internal client and the proxy server. Internet requests go

through this circuit to the proxy server, and the proxy server delivers those requests to the Internet after changing the IP address. External users only see the IP address of the proxy server. Responses are then received by the proxy server and sent back through the circuit to the client. While traffic is allowed through, external systems never see the internal systems. This type of connection is often used to connect "trusted" internal users to the Internet. (Siyan/Ware, 1995)

E. APPLICATION-LEVEL GATEWAY

An application-level proxy server provides all the basic proxy features and also provides extensive packet analysis. When packets from the outside arrive at the gateway, they are examined and evaluated to determine if the security policy allows the packet to enter into the internal network. Not only does the server evaluate IP addresses, it also looks at the data in the packets to stop hackers from hiding information in the packets.

A typical application-level gateway can provide proxy services for applications and protocols like Telnet, FTP (file transfers), HTTP (Web services), and SMTP (e-mail). A separate proxy must be installed for each application-level service. Care must be taken when evaluating possible proxy server software as some vendors achieve their "security" by providing proxies for some, but not all, services. With proxies, security policies can be much more powerful and flexible because all of the information in packets can be used by administrators to write the rules that determine how packets are handled by the gateway. It is easy to audit just about everything that happens on the gateway. You can also strip computer names to hide internal systems, and you can evaluate the contents of

packets for *appropriateness* and security. Appropriateness is an option that many organizations are looking closer at today. For instance, the system administrator has the capability to set up a filter that discards any e-mail messages that contain vulgar words. (Siyan/Ware, 1995)

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SECURITY CONSIDERATIONS

A. STATEFUL INSPECTION TECHNIQUES

One of the problems with proxies is that they must evaluate a lot of information in a lot of packets. This slows down traffic and operations significantly. In addition, you need to install a separate proxy for each application you want to support. This affects performance and increases costs. A new class of firewall product is emerging that uses stateful inspection techniques. Instead of examining the contents of each packet, the bit patterns of the packets are compared to packets that are already known to be "trusted".

For example, if you access some outside service, the server remembers things about your original request like port number, and source and destination address. This "remembering" is called *saving the state*. When the outside system responds to your request, the firewall server compares the received packets with the saved state to determine if they should be allowed in.

While stateful inspection provides speed and transparency, one of its biggest disadvantages is that inside packets make their way to the outside network, thus exposing internal IP addresses to potential hackers. This is a very undesirable feature. To avoid the potential for exploitation of one's internal IP addresses, some firewall vendors are using a combination of stateful inspection and proxies together for added security.

The debate over whether proxies or stateful inspection techniques are better is as old as firewalls and continues to be a point of debate. (Cheswick, 1994)

B. SECURITY POLICIES

No firewall can protect against inadequate or mismanaged policies. If a password gets out because a user did not properly protect it, the security of the entire network is at risk. If an internal user dials out through an unauthorized connection, an attacker could subvert the network through this "backdoor". Therefore, the system administrator must implement a "firewall policy". Obviously, the firewall and the firewall policy are two distinct things that require their own planning and implementation. A weakness in the policy or the inability to enforce the policy will weaken any protection provided by even the best firewalls. If internal users find the policy too restrictive, they may go around it by connecting to the Internet through a personal modem. In this case, the firewall becomes useless. You may not even know your systems are under attack because the firewall is guarding the wrong entrance. An example of the most basic firewall policy is as follows:

- *Block all traffic, then allow specific services on a case-by-case basis.*

This policy is restrictive but secure. However, it may be so restrictive that users circumvent it. In addition, the more restrictive your policy, the harder it will be to manage connections that are to be allowed. On screening routers, you'll need to implement complicated sets of rules, which is a difficult task. Most firewall products simplify this process by using graphical interfaces and a more efficient set of rules.

Security policies must be outlined, propagated and discussed in advance so administrators and users know what type of activities are allowed on the network. The

system administrator's policy statement should address internal and external access, remote user access, virus protection and avoidance, encryption requirements, program usage, and a number of other considerations, as outlined here:

- Network traffic to and from outside networks such as the Internet must pass through the firewall. The traffic must be filtered to allow only authorized packets to pass.
- Never use a firewall for general-purpose file storage or to run programs, except for those required by the firewall. Do not run any services on the firewall except those specifically required to provide firewall services. Consider the firewall expendable in case of an attack.
- Do not allow any passwords or internal addresses to cross the firewall.
- If you need to provide services to the public, put them on the outside of the firewall and implement internal settings that protect the server from attacks that would deny service.
- Accept the fact that you might need to completely restore public systems from backup in the event of an attack. You can implement a replication scheme that automatically copies information to a public server over a secure channel, as discussed at the end of this chapter.

For outbound connections, implement an encryption scheme to hide transmitted information. If users are accessing the Web with Web browsers, you can implement Web client-server security protocols and encryption techniques.

The system administrator also needs to evaluate what kind of traffic he or she wants to allow in from the external side of the network. Electronic mail is the usual requirement. (Ziegler, 2000)

C. SERVICE PORTS: KEY ACCESS POINTS

Service ports identify the programs and individual sessions or connections taking place on your computer. They are the numeric representations for the different network-based services and are recognized by their designations as the endpoints of a particular service connection. Server programs, otherwise known as daemons, listen for incoming connections on a service port assigned to them. By historical convention, their numerical assignment has been coordinated by the Internet Assigned Numbers Authority (IANA) and has ranged between 1 and 1023. The lower ranged ports are called "privileged" ports because programs running with system-level (i.e. superuser or root) privileges own them. Concerns arise from the fact that, while the client program may appear to have connected to the service it is intended to, this may not be the case. One can never be sure that a remote machine or service is who or what it claims to be. Higher port numbers from 1024 to 65535 are called "unprivileged" ports. They are dynamically assigned to the client end of a connection. The combination of client and server port numbered pairs, the transport protocol, along with their respective IP addresses, uniquely identify the connection. (Ziegler, 2000)

The reason service ports are of concern to the system administrator is that this is where the hacker will seek access to an internal network through a firewall. A discussion of some common ports and their function follows below.

Port Name	Port Number	Protocol\Alias
ftp	21/tcp	
telnet	23/tcp	
smtp	25/tcp	mail
whois	43/tcp	nickname
domain	53/tcp	nameserver
domain	53/udp	nameserver
finger	79/tcp	
pop-3	110/tcp	
nntp	119/tcp	readnews
www	80/tcp	http
auth	113/tcp	ident
ntp	123/udp	
https	443/tcp	

Table 2. Service Name-to-Port Mappings (Cheswick, 1994)

THIS PAGE INTENTIONALLY LEFT BLANK

V. PACKETS: IP NETWORK MESSAGES

The term packet refers to an Internet Protocol (IP) network message. The particular IP standard defines the structure of the message sent between the two computers over the network. The packet contains an information header and message body containing the data being transferred. The IP firewall (IPFW) mechanism included in Linux supports three popular message types: ICMP, UDP, and TCP. (Ziegler, 2000)

An Internet Control Message Protocol (ICMP) packet is a network-level, IP control and status message. At the network level, ICMP messages contain information about the communication between the two endpoint computers.

A User Datagram Protocol (UDP) IP packet carries UDP transport-level data between two network-based programs, without any guarantees regarding successful delivery or packet delivery ordering. Sending a UDP packet is akin to sending a postcard to another program.

A Transmission Control Protocol (TCP) IP packet carries TCP transport-level data between two network-based programs, as well, but the packet header contains additional state information for maintaining an ongoing, reliable connection. Sending a TCP packet is akin to carrying on a phone conversation with another program. Most Internet network services use the TCP communication protocol rather than the UDP. In other words, most Internet services are based on the idea of an ongoing connection with

two-way communication between a client program and a server program. (Stallings, 1995)

Version	IHL	Type of Service	Total Length
Identification	Flags	Fragment Offset	
Time to Live	Protocol	Header Checksum	
Source Address			
Destination Address			
Options + Padding			

Table 3. IP Header

A. INTERNET CONTROL MESSAGE PROTOCOL (ICMP)

An example of an ICMP message is an error status message. If you attempted to connect to a remote Web server, and a Web server wasn't running on the remote host, the remote host would return an ICMP error message indicating that the service didn't exist.

ICMP provides a means for transferring messages from routers and other hosts to a host. In essence, ICMP provides feedback about problems in the communication environment. For example, when a datagram can't reach its destination, when the router doesn't have the buffering capacity to forward a datagram, and when the router can direct the station to send traffic on a shorter route. In most cases, the ICMP is sent in response to a datagram, either by a router along the datagram's path, or by the intended destination

host. Because ICMP messages are transmitted in IP datagrams, their delivery is not guaranteed and their use cannot be considered reliable. All ICMP messages start with a 64 bit header consisting of the following:

- Type (8 bits) Specifies the type of ICMP message.
- Code (8 bits) Used to specify parameters of the message that can be encoded in one or a few bits.
- Checksum (16 bits) Checksum of the entire ICMP message. This is the same checksum algorithm used for IP.
- Parameters (32 bits) Used to specify more lengthy parameters.

In most cases, the ICMP refers to a prior datagram, the information field includes the entire IP header plus the first 64 bits of the data field of the original datagram. This enables the source host to match the incoming ICMP message with the prior datagram. The reason for including the first 64 bits of the data field is that this will enable the IP module in the host to determine which upper-level protocol or protocols were involved. In particular, the first 64 bits would include a portion of the TCP header or other transport-level header. ICMP messages include the following:

- Destination unreachable
- Time exceeded
- Parameter problem
- Source quench
- Redirect
- Echo
- Echo reply
- Timestamp
- Timestamp reply

The "*destination unreachable*" message covers a number of contingencies. A

router may return this message if it doesn't know how to reach the destination network. In some networks, an attached router may be able to determine if a particular host is unreachable, and then return the message. The destination host itself may return this message if the user protocol or some higher-level service access point is unreachable. This could happen if the corresponding field in the IP header was set incorrectly. If the datagram specifies a source route that is unusable, a message is returned. Finally, if a router must fragment a datagram but the Don't Fragment flag is set, a message is returned.

Type	Code	Checksum
Unused		
IP Header + 64 bits of original datagram		

Table 4. Destination Unreachable/Time Exceeded/Source Quench Header

A router will return a "*time-exceeded*" message if the lifetime of the datagram expires. A host may send this message if it cannot complete reassembly within a time limit.

A syntactic or semantic error in an IP header will cause a "*parameter problem*" message to be returned by a router or host. For example, an incorrect argument may be provided with an option. The parameter field contains a pointer to the octet in the original header where the error was detected.

Type	Code	Checksum
Pointer	Unused	
IP Header + 64 bits of original datagram		

Table 5. Parameter Problem Header

The "*source quench*" message provides a rudimentary form of flow control. Either a router or a destination host may send this message to a source host, requesting that it reduce the rate at which it is sending traffic to the internet destination. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages; this message can be used by a router or host that must discard datagrams because of a full buffer. In this case, the router or host will issue a source quench message for every datagram that it discards. In addition, a system may anticipate congestion and issue such messages when it buffers approach capacity. In that case, the datagram referred to in the source quench message may well be delivered. Thus, receipt of the message doesn't imply delivery or nondelivery of the corresponding datagram.

A router sends a "*redirect*" message to a host on a directly connected router to advise the host of a better route to a particular destination; the following is an example of its use. A router, R1, receives a datagram from a host on a network to which the router is attached. The router, R1, checks its routing table and obtains the address for the next router, R2, on the route to the datagram's destination network. If R2 and the host identified by the internet source address of the datagram are on the same network, a

redirect message is sent to the host. The redirect message advises the host to send its traffic for the specified network directly to R2, as this is a shorter path to the destination. The router forwards the original datagram to its internet destination (via R2). The address of R2 is contained in the parameter field of the redirect message.

Type	Code	Checksum
Gateway Internet Access		
IP Header + 64 bits of original datagram		

Table 6. Redirect Header

The "*echo*" and "*echo reply*" messages provide a mechanism for testing that communication is possible between network entities. The recipient of an echo message is obligated to send an echo reply message. An identifier and sequence number are associated with the echo message to be matched in the echo reply message. The identifier might be used like a service access point to identify a particular session, and the session, and the sequence number might be incremented on each echo request sent.

Type	Code	Checksum
Identifier	Sequence Number	
IP Header + 64 bits of original datagram		

Table 7. Echo and Echo Reply Header

The "timestamp" and "timestamp reply" messages provide a mechanism for sampling the delay characteristics of the internet. The sender of a timestamp message

may include an identifier and a sequence number in the parameters field and include the time that the message is sent. The receiver records the time it received the message and the time that it transmits the reply message in the timestamp reply message. If the timestamp message is sent using strict source routing, then the delay characteristics of a particular route can be measured. (Stallings, 1997)

Type	Code	Checksum
Identifier	Sequence Number	
Originate Timestamp		

Table 8. Timestamp Header

Type	Code	Checksum
Identifier	Sequence Number	
Originate Timestamp		
Receive Timestamp		
Transmit Timestamp		

Table 9. Timestamp Reply Header

B. USER DATAGRAM PROTOCOL (UDP)

UDP is "stateless" meaning that it is connectionless. It is also an "unreliable" transport delivery protocol – indicating not that it isn't useful, but that it doesn't employ any of the verification services other protocols have. A program sends a UDP message, which may or may not be received or responded to. No acknowledgment of receipt is returned or expected. No flow control is provided, so a UDP datagram is silently dropped if it can't be processed along the way. In general, most "real time" services like Real Audio use UDP. (Stallings, 1997)

Source Port	Destination Port
Length	Checksum

Table 10. UDP Header

C. TRANSMISSION CONTROL PROTOCOL (TCP)

Unlike UDP, TCP is a reliable protocol in that it employs a system to verify that each packet sent was received as sent. The system uses a three-way handshake characterized by the exchange of a series of synchronization and acknowledgment flags. The clients initial message contains the SYN flag along with a synchronization sequence number. When the packet is received at the server machine, it is sent to the appropriate service port, where a port socket pair is established. The server machine then responds with an ACK flag, acknowledging the SYN message, along with its own SYN (synchronization request). (Stallings, 1997) This synchronization request is then

acknowledged and the TCP connection is established for transmitting data in both directions. A similar "handshake" action is used at the end of a session to tear down the connection.

<table>
<tr><td colspan="5">Source Port</td><td colspan="3">Destination Port</td></tr>
<tr><td colspan="8">Sequence Number</td></tr>
<tr><td colspan="8">Acknowledgement Number</td></tr>
<tr><td>Data Offset</td><td>Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window</td></tr>
<tr><td colspan="5">Checksum</td><td colspan="3">Urgent Pointer</td></tr>
<tr><td colspan="8">Options + Padding</td></tr>
</table>

Table 11. TCP Header

THIS PAGE INTENTIONALLY LEFT BLANK

VI. TYPES OF ATTACK

Normally, the route a packet takes from its source to its destination is determined by the routers between the source and destination. The packet itself only says where it wants to go (the destination address), and nothing about how it expects to get there. There is an optional way for the sender of a packet (the source) to include information in the packet that tells the route the packet should get to its destination; thus the name "source routing". For a firewall, source routing is noteworthy, since an attacker can generate traffic claiming to be from a system "inside" the firewall. In general, such traffic wouldn't route to the firewall properly, but with the source routing option, all the routers between the attacker's machine and the target will return traffic along the reverse path of the source route. Implementing such an attack is quite easy; so firewall builders should not discount it as unlikely to happen.

In practice, source routing is very little used. In fact, generally the main legitimate use is in debugging network problems or routing traffic over specific links for congestion control for specialized situations. When building a firewall, source routing should be blocked at some point. Most commercial routers incorporate the ability to block source routing specifically, and many versions of Unix that might be used to build firewall bastion hosts have the ability to disable or ignore source routed traffic. (Cheswick, 1994)

A. ICMP REDIRECTS AND BOMBS

An ICMP Redirect tells the recipient system to over-ride something in its routing table. It is legitimately used by routers to tell hosts that the host is using a non-optimal or defunct route to a particular destination, i.e. the host is sending it to the wrong router. The wrong router sends the host back an ICMP Redirect packet that tells the host what the correct route should be. If you can forge ICMP Redirect packets, and if your target host pays attention to them, you can alter the routing tables on the host and possibly subvert the security of the host by causing traffic to flow via a path the network manager didn't intend. ICMP Redirects also may be employed for denial of service attacks, where a host is sent a route that loses it connectivity, or is sent an ICMP Network Unreachable packet telling it that it can no longer access a particular network.

Many firewall builders screen ICMP traffic from their network, since it limits the ability of outsiders to ping hosts, or modify their routing tables. (Cheswick, 1994)

B. DENIAL OF SERVICE

Denial of service is when someone decides to make your network or firewall useless by disrupting it, crashing it, jamming it, or flooding it. The problem with denial of service on the Internet is that it is impossible to prevent. The reason has to do with the distributed nature of the network: every network node is connected via other networks, which in turn connect to other networks, etc. A firewall administrator or ISP only has control of a few of the local elements within reach. An attacker can always disrupt a connection "upstream" from where the victim controls it. In other words, if someone

wanted to take a network off the air, they could do it either by taking the network off the air, or by taking the networks it connects to off the air, ad infinitum. There are many, many, ways someone can deny service, ranging from the complex to the brute-force. If you are considering using Internet for a service, which is absolutely, time or mission critical, you should consider your fall-back position in the event that the network is down or damaged. (Cheswick, 1994)

C. SMTP SESSION HIJACKING

This is where a spammer will take many thousands of copies of a message and send it to a huge list of email addresses. Because these lists are often so bad, and in order to increase the speed of operation for the spammer, many have resorted to simply sending all of their mail to an SMTP server that will take care of actually delivering the mail. Of course, all of the bounces, spam complaints, hate mail, and bad PR come for the site that was used as a relay. There is a very real cost associated with this, mostly in paying people to clean up the mess afterward. (Cheswick, 1994)

D. EXPLOITING BUGS IN APPLICATION

Various versions of web servers, mail servers, and other Internet service software contain bugs that allow remote (Internet) users to do things ranging from gain control of the machine to making that application crash and just about everything in between. The exposure to this risk can be reduced by running only necessary services, keeping up to date on patches, and using products that have been around a while. (Cheswick, 1994)

E. BUGS IN OPERATING SYSTEMS

Again these are typically initiated by users remotely. Operating systems that are relatively new to IP networking tend to be more problematic, as more mature operating systems have had time to find and eliminate their bugs. An attacker can often make the target equipment continuously reboot, crash, lose the ability to talk to the network, or replace files on the machine.

Here, running as few operating system services as possible can help. Also, having a packet filter in front of the operating system can reduce the exposure to a large number of these types of attacks.

And, of course, choosing a stable operating system will help here as well. When selecting an OS, don't be fooled into believing that "the pricier, the better". Free operating systems are often much more robust than their commercial counterparts. (Cheswick, 1994)

F. SYN ATTACK

A new threat emerged for Internet-connected systems called the *SYN attack.* In such an attack, a malicious person floods a server with session-request packets. The server tries to establish a session for each of those requests, but the malicious user makes sure that a response is never sent to the server after the initial request. It's like someone reaching out to shake your hand, then pulling it away when you reach out with your hand. The server keeps waiting to "shake hands" with the hacker's system and eventually

crashes when its runs out of resources to handle the load. The hacker has caused a *denial-of-service* attack in which legitimate users cannot access the system.

This type of attack was successfully staged against Panix, a New York Internet Service Provider in September of 1996. Thousands of people were denied Internet access at the time, and similar attacks took place elsewhere, apparently after the attack strategy was discussed openly on the Internet. While no data was destroyed, this incident points out how vulnerable our information systems and networks are. In many cases, we just don't know all of the vulnerabilities. (Cheswick, 1994)

THIS PAGE INTENTIONALLY LEFT BLANK

VII. RESEARCH

A. APPROACH

The foundation for this research was established by first becoming familiar with the software at hand. More specifically, a study of Linux was conducted by building a comprehensive library of reference books having to do with Linux, Network Security, Linux Security and even Linux Firewalls. In logical progression, a study of current hacking techniques, hacker software and hacking web sites was undertaken in an attempt to learn the process that would go into penetrating the network. Understanding network mapping, port scanning and Trojan horse penetration is key to understanding how to implement the measures deployed to block such invasive probes and attacks.

As the research was carried out, previous incarnations of Linux-based firewall and network access control products came to light. TCP Wrappers, a host-based solution, appeared as the first true answer to network access control in Linux. While TCP Wrappers is not a firewall, it adds network access control through a simple, but reliable mechanism. It is the closest one could get to firewall functionality without actually deploying a full-scale packet filter and it's a good alternative when one can't use a firewall but still needs network access control. It has the added advantage of connection logging.

Still, there seemed to be a need for something more comprehensive. So, ipfwadm was created to have both firewall functionality and packet filtering characteristics. It is

used to set up, maintain, and inspect the IP firewall and accounting rules in the Linux kernel. These rules can be divided up into four different categories: accounting of IP packets, the IP input firewall, the IP output firewall, and the IP forwarding firewall. For each of these categories, a separate list of rules is maintained. For such a small tool, ipfwadm is a formidable personal firewall solution. Ipfwadm was inherent in all 2.2 and earlier kernel packages. It has since been superceded by ipchains, in the 2.2 and beyond kernel package. The primary difference, from a operator's standpoint, is that commands are now in uppercase while arguments are in lowercase. There are other differences, but in all other respects, ipchains works much like ipfwadm. (Ziegler, 2000)

B. IPCHAINS

It turns out that ipchains is a packet filtering firewall, whose operation is similar to the packet filtering example discussed above. As with ipfwadm, ipchains' operation is based on a set of system administrator defined rules; input, output, forward, and user defined rules. Ipchains is used to setup, maintain, and inspect the IP firewall rules in a Linux kernel.

Ipchains understands the following general rules; ACCEPT, DENY, REJECT, MASQ, REDIRECT and RETURN. As a packet is received and processed, it is compared against the first rule. If the packet does not match, then the next rule in the chain is checked; if it does not match either, the process continues until all the rules are exhausted. At this point, it utilizes the default rule.

Of course, one of the most important aspects of defining firewall rules is that the order in which the rules are defined is important. Packet filtering rules are stored in kernel tables, in an input, output or forward chain, in the order in which they are defined. Individual rules are inserted at the beginning of the chain or appended to the end of the chain. The order the rules are defined in is the order they'll be added to the kernel tables, and thereby the order in which the rules will be compared against each packet.

As each externally originating packet arrives at a network interface, its header fields are compared against each rule in the interface's input chain until a match is found. Conversely, as each internally originating packet is sent to a network interface, its header fields are compared against each rule in the interface's output chain until a match is found. In either direction, when a match is found, the comparison stops and the rule's packet disposition is applied: ACCEPT, REJECT or DENY. If the packet doesn't match any rule on the chain, the default policy for the chain is applied. The bottom line is that the first matching rule "wins". (Ziegler, 2000)

Input chains regulate the acceptance of incoming IP packets. Packets coming in via one of the local network interfaces are checked against the input firewall rules. Output chains regulate the outgoing IP packets. All packets that are ready to be sent via one of the local network interfaces are run against the output firewall rules. Forwarding chains require packets that are sent requiring to be routed through the firewall to be checked against the forwarding firewall rules. User defined chains can be called within a chain based on a packet that match criteria within another chain. (Schwoerer, 2000)

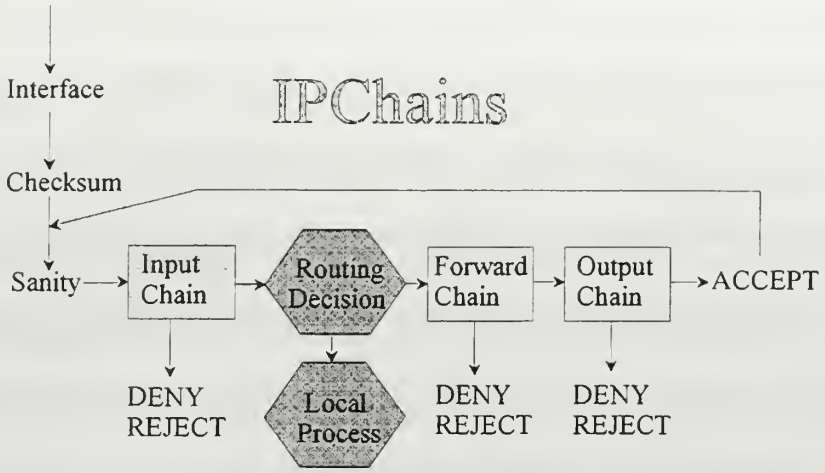

Figure 2. IPChains Processing Diagram

C. A TYPICAL ATTACK

If one was to conduct a typical attack, it would be carried out systematically using a series of readily available hacking tools that can be easily downloaded for free from the Internet. The attack would begin with a generic scan of the targeted IP address or range of addresses. The scan would provide results identifying all assets by IP address that appear to be "up" or active at that time. Next, the hacker would choose the address of the victim he wanted to exploit and run a series of port scans on it to determine the possibility of the victim's vulnerabilities. Using this information, the hacker would then attempt to penetrate the victim computer via one of the vulnerable ports identified by the port scan. Of course, this is a very general hacking case that *could* lead to an attack, however, not every penetration is an *attack*. Many times, the hackers are just fooling around and your computer or network may be the unwitting victim of a silly game.

Either way, many, many different scenarios exist for the exploitation of victim network assets. Everyone would like to ensure the security of his or her own systems regardless of the attacker's motivation. This is why it is imperative to have a strong understanding of the OSI model, the three most common protocols, TCP, ICMP, and UDP service ports and how a hacker can use his or her knowledge of these concepts to exploit inherent vulnerabilities in the system.

D. HACKING TOOLS

1. NMAP

One very important, easy to use tool available on the Internet is a program called NMAP. NMAP does three things. First, it will ping a number of hosts to determine if they are "alive" or not. Second, it will portscan hosts to determine what services are listening. Third, it will attempt to determine the OS of hosts (either Windows/NT or UNIX)

Of course NMAP is very configurable, and any of these steps may be omitted, (although portscanning is necessary in order to do an OS scan), and there are multiple ways to accomplish most of these, and many command line switches to tweak the way that NMAP operates.

a) TARGET SELECTION

You can specify NMAP targets both on the command line or give a list of targets in a filename with the -i option. As the NMAP help documentation suggests you

can use the hostname/mask method of specifying a range of hosts (cert.org/24 or 192.88.209.5/24) or you can give a explicit IP range (192.88.209.0-255). The '24' in 'cert.org/24' is the number of bits in the mask, so /32 means "just that host", /24 means "the 256 addresses in that Class C", /16 means "the 65536 addresses in that Class B", /8 would be "the 2^{24} addresses in that Class A" and /0 would scan all possible (IPv4) 2^{32} IP addresses.

b) PING SCANS

The default behavior of NMAP is to do both an ICMP ping sweep (the usual kind of ping) and a TCP port 80 ACK ping sweep. If an admin is logging these this will be fairly characteristic of NMAP. This behavior can be changed in several ways. The easiest way is, of course, to simply turn off ping sweeps with -P0.

If you want to do a standard ICMP ping sweep use -PI. If you are trying to get through a firewall, though, ICMP pings will likely be blocked and using packet filtering ICMP pings can even be dropped at the host. To get around this NMAP tries to do a TCP "ping" to see if a host is up. By default it sends an ACK to port 80 and expects to see a RST from that port if the host is up. To do only this scan and not the ICMP ping scan use -PT. To specify a different port than port 80 to scan for specify it immediately afterwards, e.g. -PT32523 will ACK ping port 32523. Picking a random high-numbered port in this way may work *much* better than the default NMAP behavior of ACK pinging port 80. This is because many packet filter rules are setup to let through all packets to high numbered ports with the ACK bit set, but sites may filter port 80 on every

machine other than their publicly accessible webservers. You can also do both an ICMP ping scan and an ACK scan to a high numbered port with, e.g. -PB32523. However, if a site has a really, really intelligent firewall that recognizes that your ACK packet isn't part of an ongoing TCP connection it might be smart enough to block it. For that reason, you may get better results with a TCP SYN sweep with -PS. In this case, scanning a high-numbered port will probably not work, and instead you need to pick a port, which is likely to get through a firewall. Port 80 is not a bad pick, but something like ssh (port 22) may be better.

So the first question to ask yourself is if you care about wasting time scanning machines which are not up and if you care about getting really complete coverage of the network? If you don't care about wasting time and really want to hit all the machines on a network, then use -P0. Pinging machines will only cause you to have more of a signature in any log files and will eliminate machines, which might possibly be up. Of course, you will waste time scanning all the IP numbers, which aren't assigned.

If you do ping machines, an ICMP ping sweep is probably more likely to be missed or ignored by system administrators. It doesn't look all that hostile. If you think you're up against a firewall you should experiment with which kinds of pings seem to get through it. Do ICMP pings work at all? Can you ping their webserver? If not, then don't bother with ICMP pings. Can you ACK ping their webserver? If not, then you have to go with SYN pings. What if all you want to do is a ping scan? Then use -sP.

c) PORT SCANNING

The most general scan is a TCP connect() scan (-sT). Since these are loggable, you probably don't want to do these. SYN scans (-sS) are the workhorse of scanning methods. They are also called "half-open" scans because you simply send a SYN packet, look for the return SYN|ACK (open) or RST (closed) packet and then you tear down the connection before sending the ACK that would normally finish the TCP 3-way handshake. These scans don't depend on the characteristics of the target TCP stack and will work anytime a connect scan would have worked. They are also harder to detect -- TCP-wrappers or anything outside of the kernel shouldn't be able to pick up these scans -- packet filters like ipfwadm, ipchains or a commercial firewall can though. If a box is being filtered NMAP's SYN scan will detect this and report ports, which are being filtered.

FIN (-sF), NULL (-sN) and XMAS (-sX) scans are all similar. They all rely on RFC-compliance and as such don't work against boxes like Win95/98/NT or IRIX. They also work by getting either a RST back (closed port) or a dropped packet (open port). Of course, the other situation where you might get back a dropped packet is if you've got a packet filter blocking access to that port. In that case you will get back a ton of false open ports. A few years back these kinds of scans might have been stealthy and undetectable. These days they probably aren't.

You can combine any of the SYN, FIN, NULL or XMAS scans with the (-f) flag to get a small fragment scan. This splits the packet, which is sent, into two tiny

fragments, which can sometimes get through firewalls and avoid detection. Unfortunately, if you're not running a recent version of an open source O/S (Linux or Net/Open/FreeBSD) then you probably can't fragment scan due to the implementation of SOCK_RAW on most Unix (Solaris, SunOS, IRIX, etc). For the initiated out there, you could modify libpcap to allow you to send packets in addition to sniffing them by opening the packet capture device rw instead of ro. Then you need to build a link-layer (probably Ethernet) header and then you could implement your own fragmentation scanner. For bonus points implement all of the different SYN, FIN, NULL and XMAS scans and allow for sending the fragments out in reverse order (which helps for getting through firewalls). This hasn't been done (yet) in NMAP due to the fact that NMAP needs to support multiple different link layer interfaces (not just Ethernet) and needs code for dealing with ARP.

UDP scanning (-sU) in NMAP has the same problem as FIN scans in that packet filtered ports will turn up as being open ports. It also runs extremely slowly against machines with UDP packet filters.

Another type of scan is the bounce scan (-b) which, if there is insufficient logging on the ftp host you're using to bounce, is completely untraceable. Recent FTP servers shouldn't let you do these kinds of scans.

The last scanning option worth mentioning is identd scanning (-I), which only works with TCP connect scans (-sT). This will let you know the owner of the daemon, which is listening on the port. Provided, of course, that the site is running identd

and is not doing something intelligent like using a cryptographic hash (i.e. pidentd -C). You have to make complete 3-way TCP handshakes for this to work, so this is not very stealthy. It does, however, give you a lot of information. It only works against machines that have port 113/auth open.

d) SOURCE IP DECEPTION

You can also take advantage of the fact that you can change your source address. The simplest way to do this is with -S. If you are on a broadcast Ethernet segment you could change your source address to an IP, which doesn't exist, and then you simply sniff the network for the reply packets. If you are not on a leaf node/network then as long as the reply packet will get routed by you, you can use it. To turn this on its head the next time you get scanned, do a traceroute on the machine that scanned you. Any of the machines on any of the networks that those packets went through could have been the machine, which was really scanning you.

The other deceptive measure is to use decoy scans. You spoof a ton of scans originating from decoy machines and insert your IP in the middle of it somewhere. The admin at the site you are scanning is presented with X number of scans and no way to determine which one actually did it. For bonus points, combine this with the previous tactic and spoof an IP address, which doesn't exist. If you don't spoof your own IP address make sure to use "likely" decoys. For instance, use machines, which were connected to the net at the time you made your scans and don't use sites like

www.microsoft.com. Ideally you want a lot of Linux boxes as decoys. The more decoys the better, but obviously the slower the scan will go.

e) OS SCANNING

This is the -O option. To use it requires one open and one closed port. The closed port is picked at random from a high-numbered port. Machines which do packet filtering on high-numbered ports will cause problems with OS detection (many sites will filter packets to high numbered ports which don't have the ACK bit set). Also excessive packet loss will cause problems with OS detection. If you run into trouble try selecting an open port which isn't being served by inetd (e.g. ssh/22 or portmap/rpcbind/111). OS scanning also reports the TCP sequence number prediction vulnerability of the system. If you're using 31337 you will be able to use this to exploit trust relationships between this machine and other machines. There's a reasonably decent phrack article on this in phrack P48-14, but you should beware that it isn't this easy -- you need to worry about ARP and if you're trying to exploit rsh/rlogin you need to worry about spoofing the authorization connection as well. (Granquist, 2000)

2. RED BUTTON

Red Button was created to demonstrate security flaws associated with the RedButton Bug in Microsoft Windows NT v 3.5x and 4.0 Operating Systems that potentially affect the majority of NT-based networks. It is a "must have" tool used for checking if a system is vulnerable to the Red Button Bug.

Red Button logs on remotely to a target computer without presenting a user name or password. It then gains access to the resources available to common users and determines the current name of the built-in administrator account. It also reads registry entries and displays the name of any registered owners and lists all shares, including hidden ones.

Why is this of concern if the firewall is Linux-based? If any of the computers on the internal network are NT-based, and the firewall can be penetrated, then their information is vulnerable and can be compromised. To be safe, block ports 135-139 at the firewall to remove the Red Button vulnerability from possible attack.

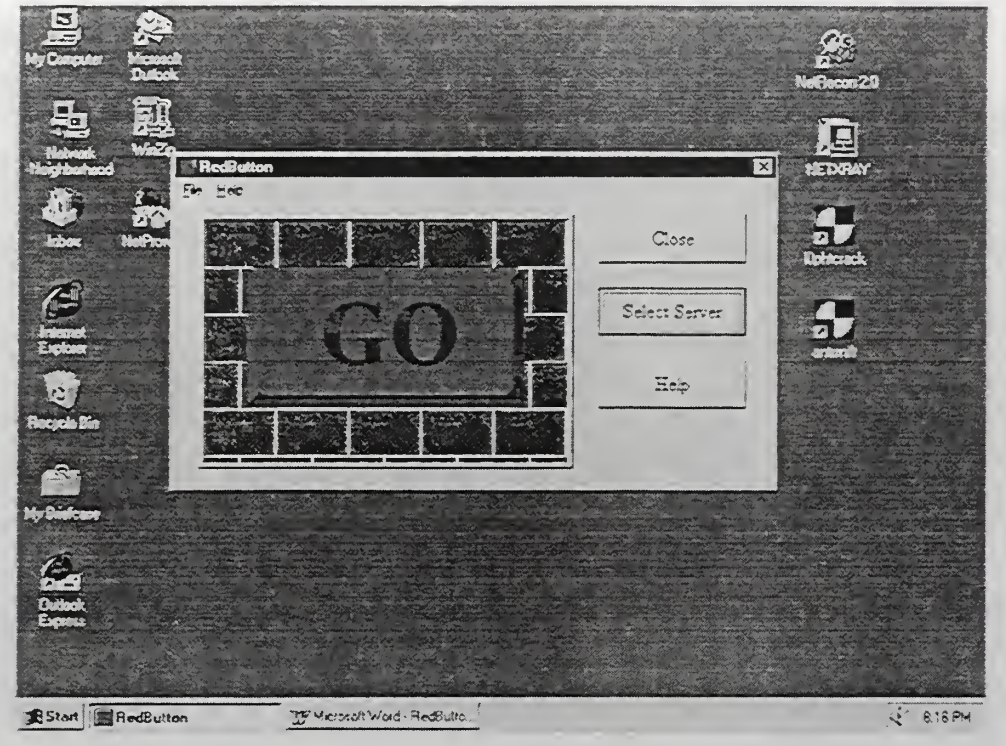

Figure 3. Red Button Hacking Tool

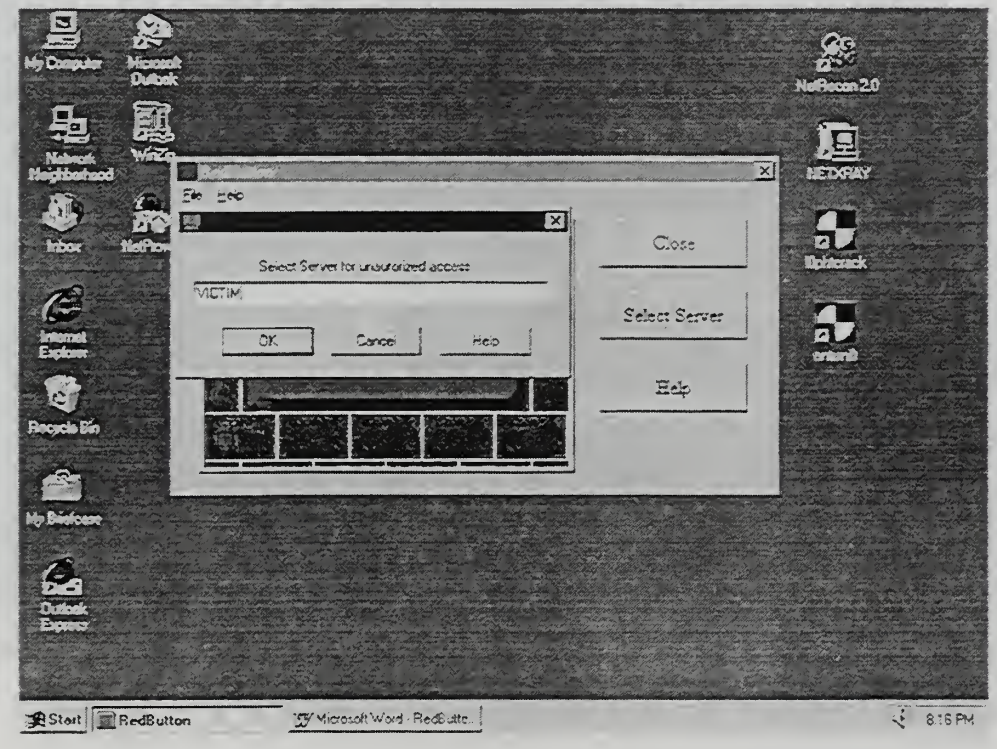

Figure 4. Selecting the Server Name in Red Button

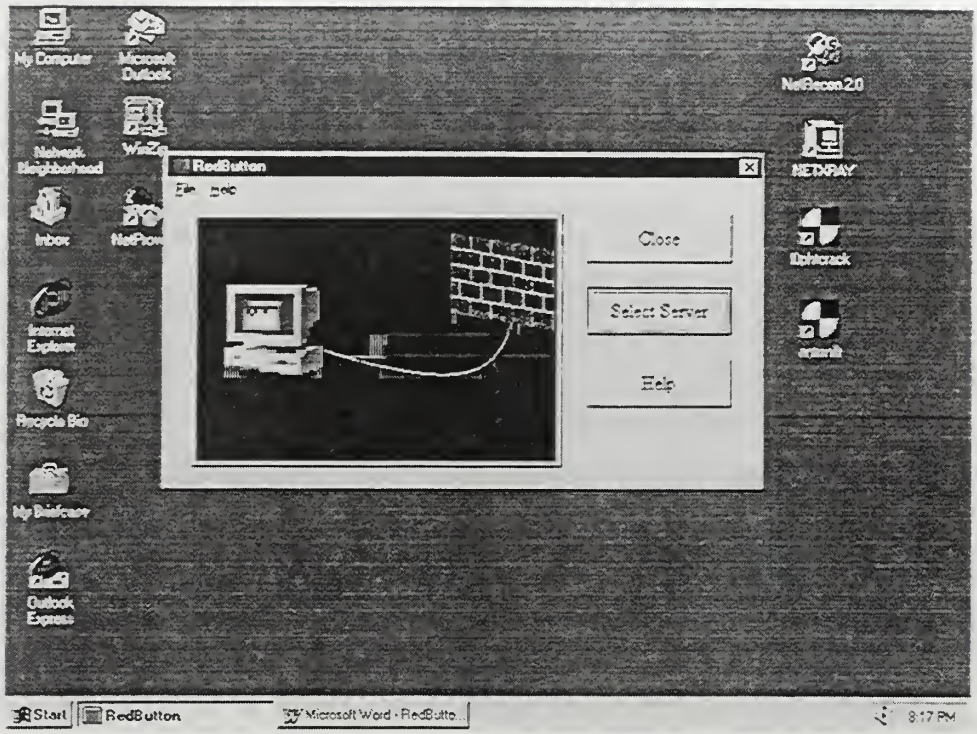

Figure 5. Attempting Connection with Red Button

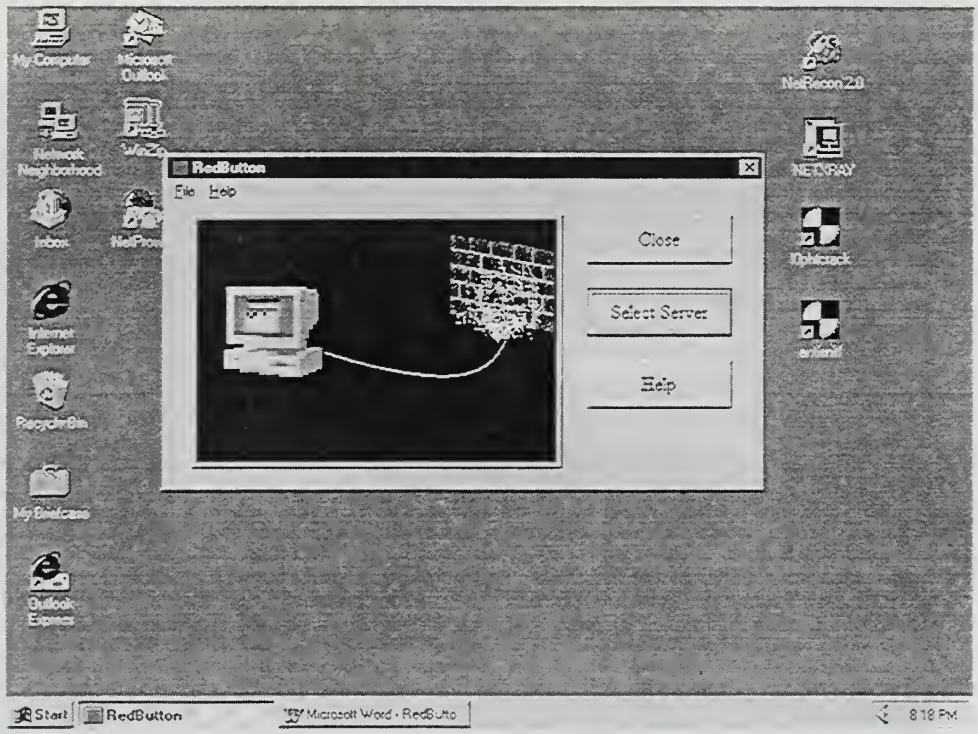

Figure 6. Breaking Through the Firewall in Red Button

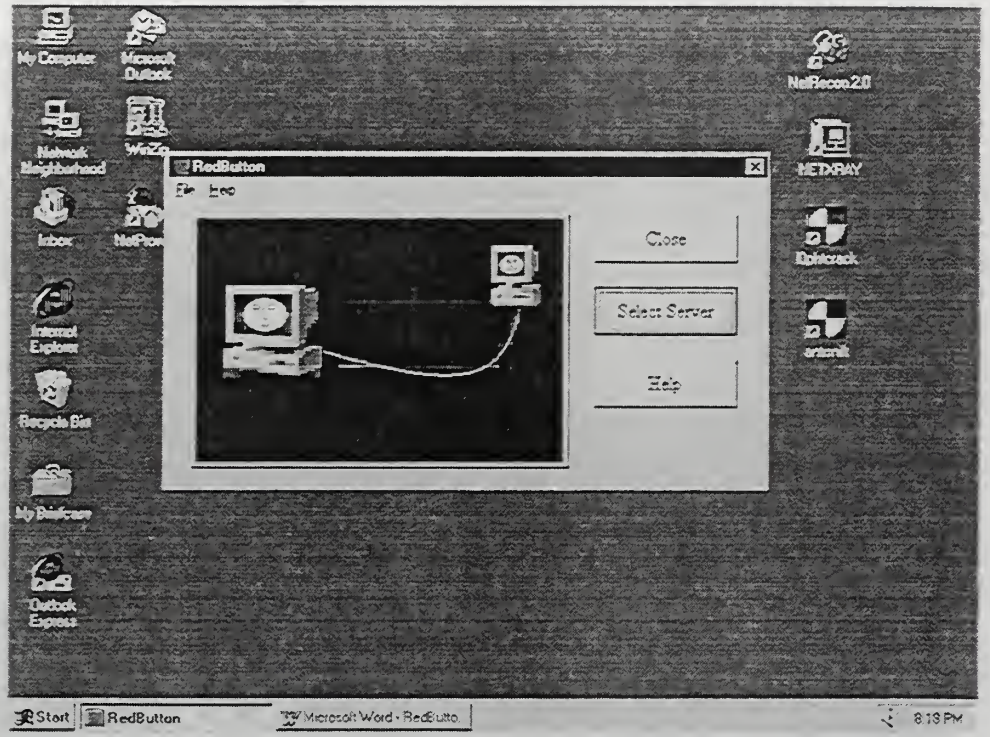

Figure 7. Successful Connection in Red Button!

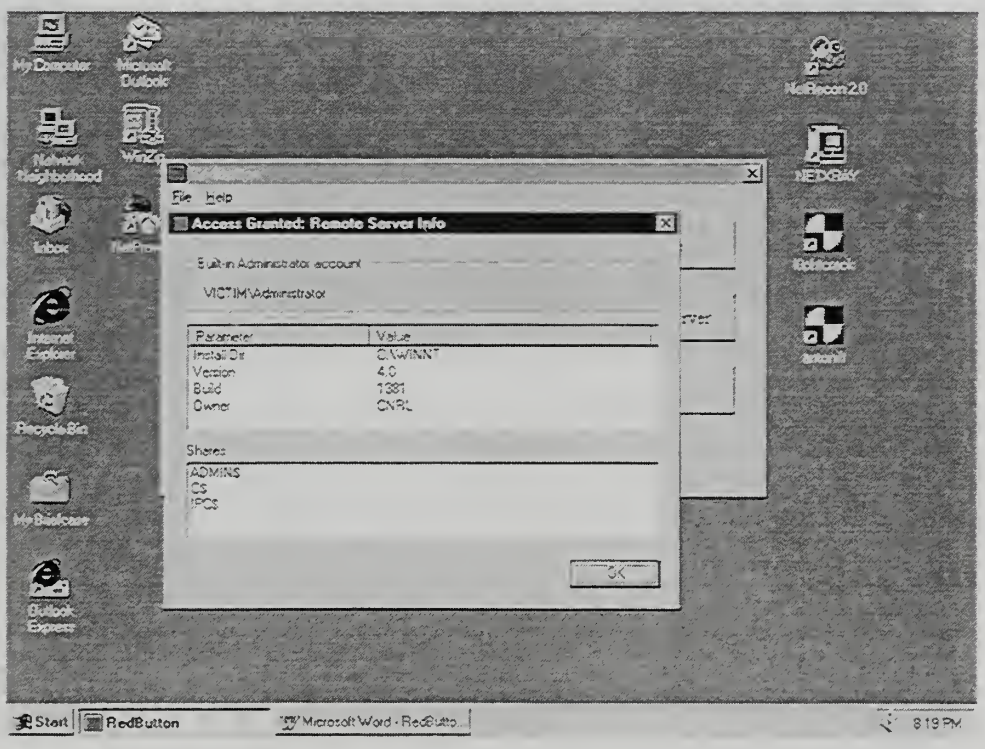

Figure 8. Results from Red Button Penetration

3. SHADOW ADVANTIS ADMINISTRATOR

Shadow is an attempt to consolidate a series of hacking tools into one user-friendly GUI-driven package. It is a very formidable tool when used by an experienced hacker.

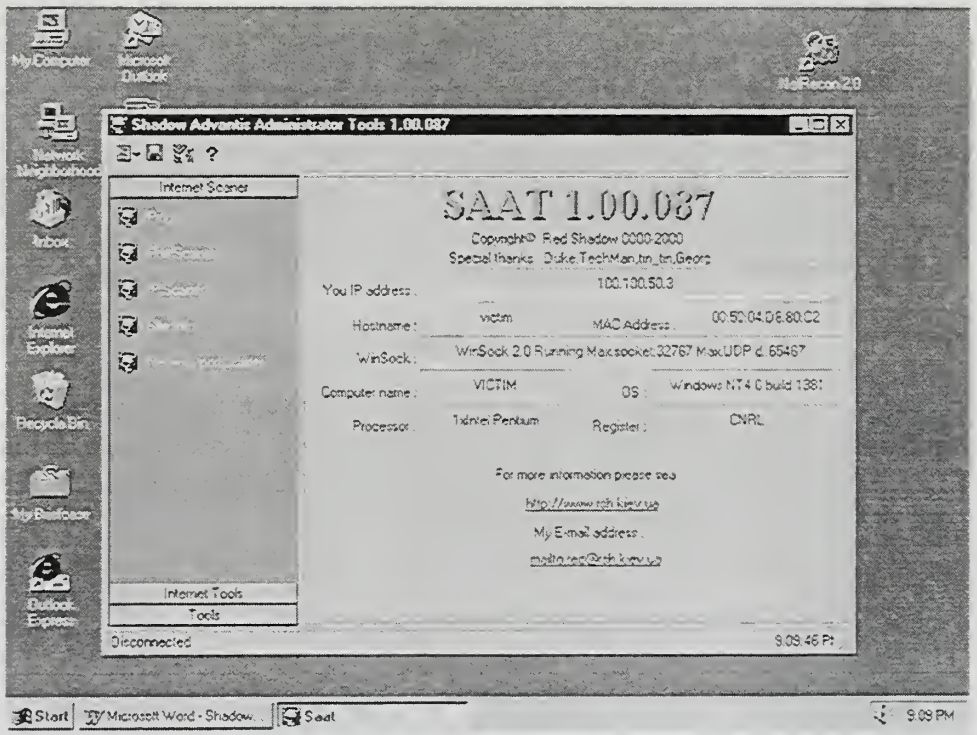

Figure 9. Shadow Advantix Administrator Tool

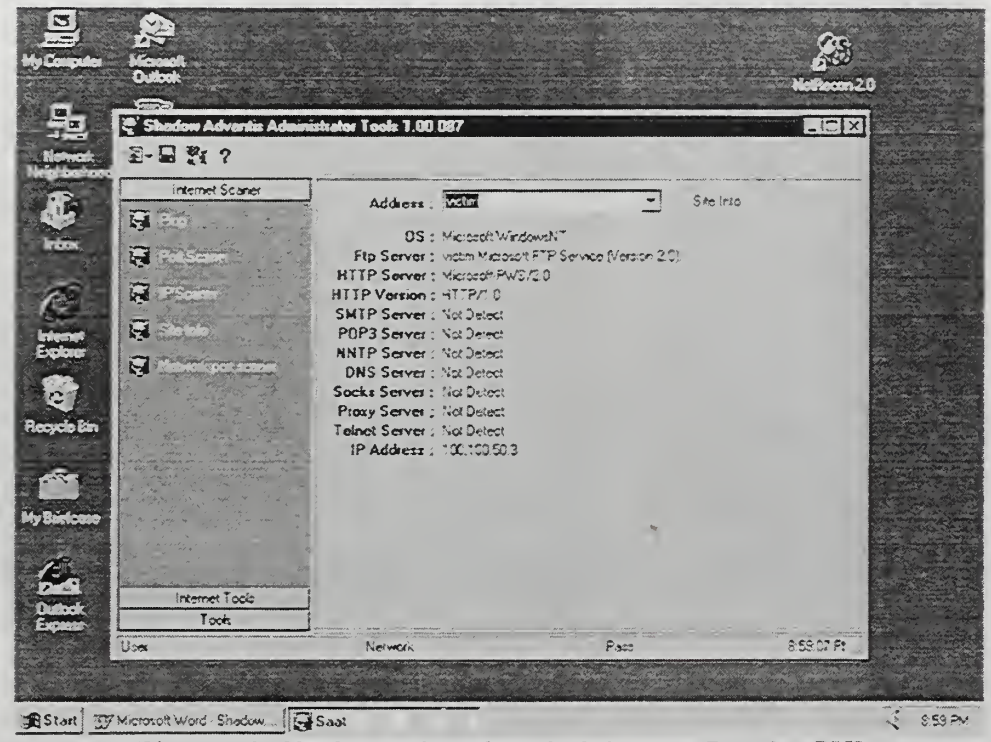

Figure 10. Shadow Advantix Administrator Results GUI

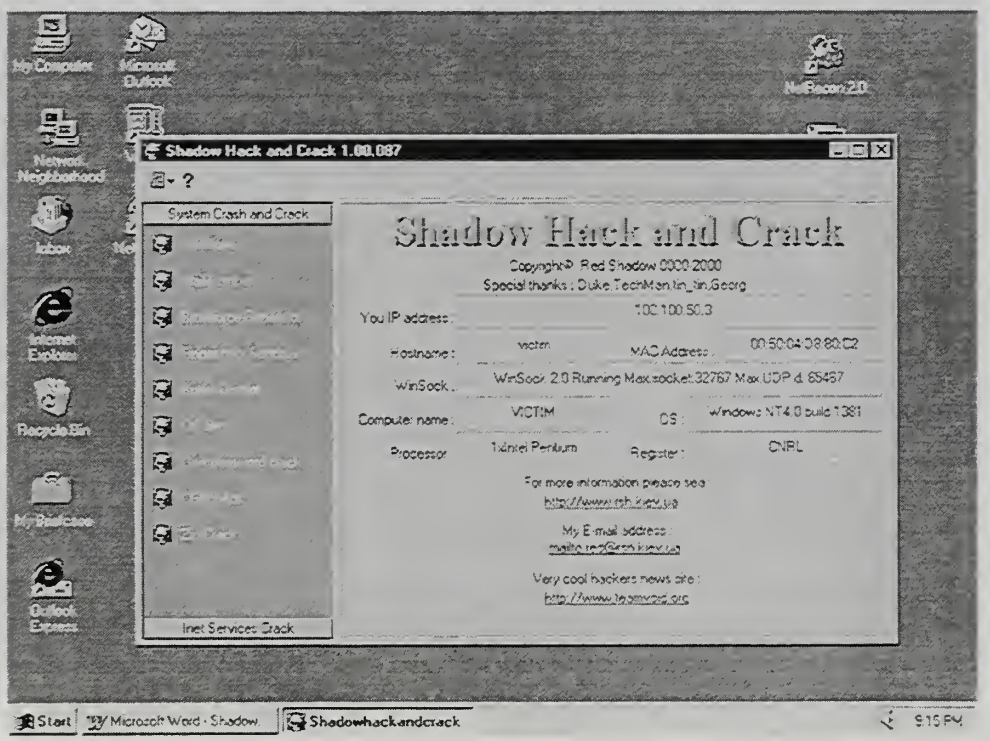

Figure 11. Shadow Hack and Crack Intro GUI

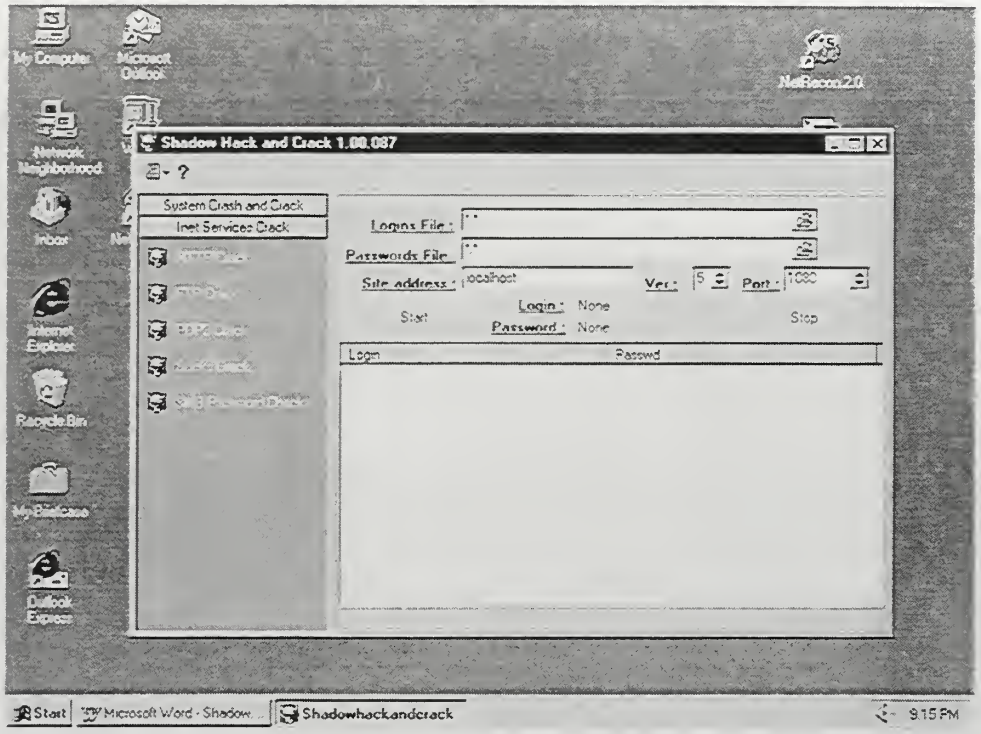

Figure 12. Shadow Hack and Crack Results GUI

4. SUB 7

Sub7 is another example of a user-friendly hacking tool designed with the unindoctrinated in mind. With powerful tools like these in one`s hands, the extent of systems at risk is astronomical. Still, remember that Sub7 is a Trojan horse that must first be delivered to the victim computer prior to being remotely activated and controlled. Generally, a Trojan horse like Sub7 would be wrapped in an executable file for delivery to the victim system via email.

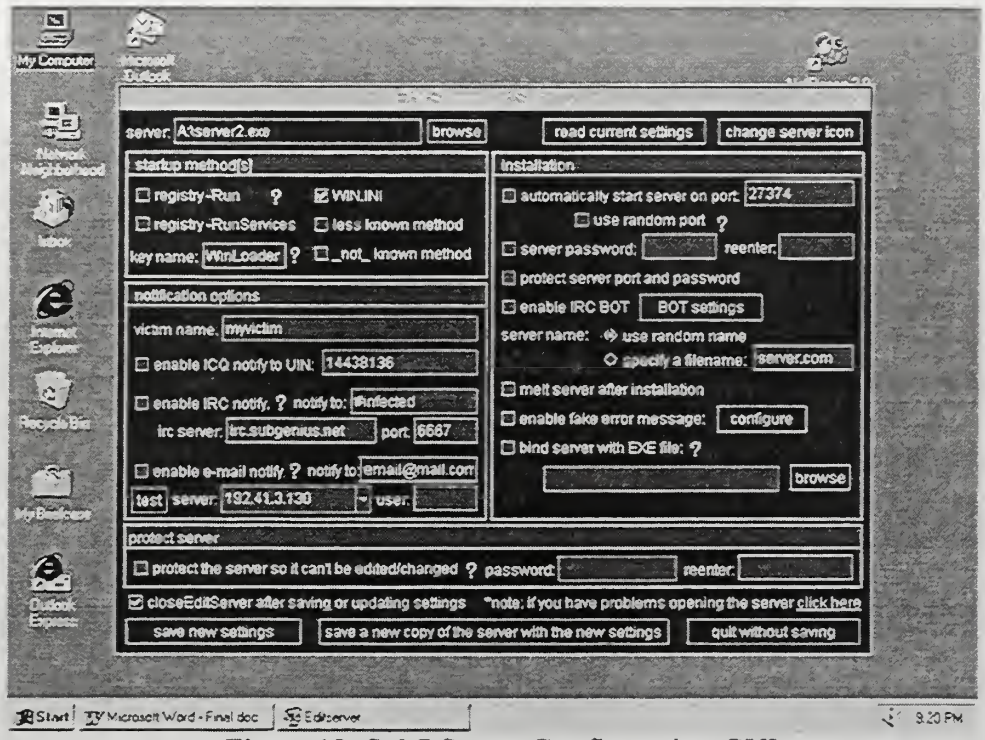

Figure 13. Sub7 Server Configuration GUI

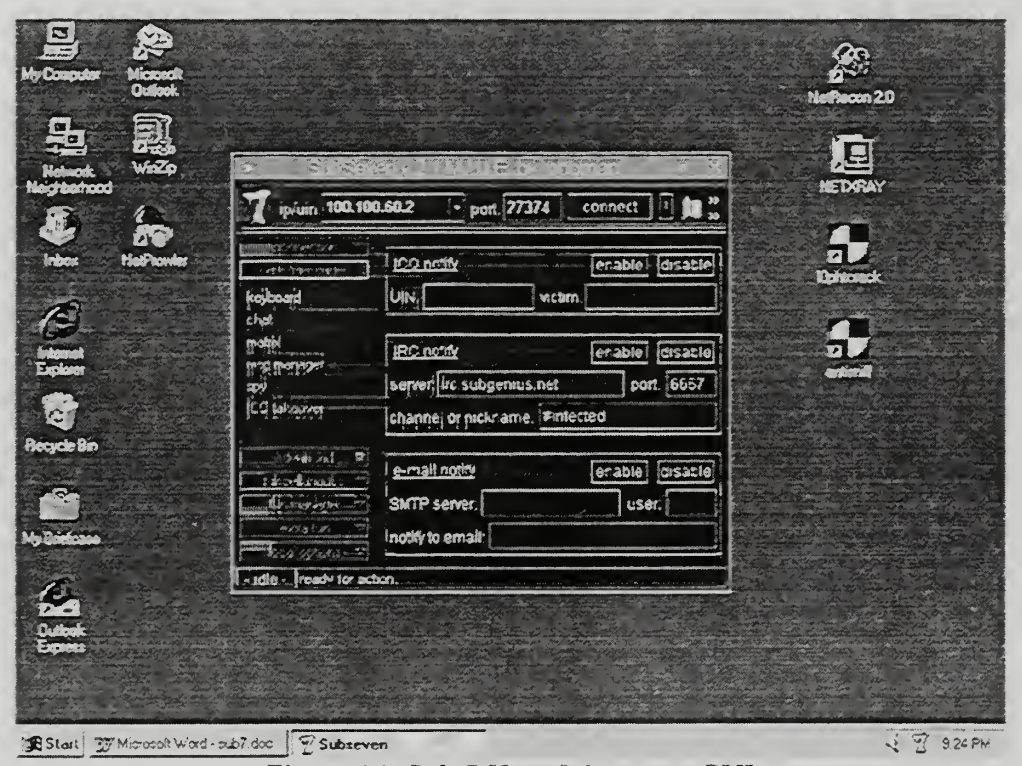

Figure 14. Sub 7 Keys/Messages GUI

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. SYSTEM CONFIGURATION

A. HARDWARE

For the purposes of what I was trying to accomplish with my research, I chose to set up a local area network (LAN) that would be akin to something used in small business applications. I chose to network four separate computers, a so-called "victim", an "attacker", the router/firewall, and a system monitor. All four computers were fully capable Pentium III, IBM compatible Micron machines, which included 64 Mbytes of Ram, a 10 Gigabyte hard drive, 3.5 floppy drive, a 100 Mbytes ZIP drive and 64X CD ROM drive.

To interconnect the LAN, I used two BlackBox© autosensing, dual speed, mini hubs. Each hub supports eight twisted pair, RJ-45 "plug and play" ports and one up-link 8X port used for cascading hubs for extended purposes. Dual speed refers to the ability of the hubs to accept systems operating at both 10Base-T (10 Mbps Ethernet IEEE 802.3) and 100Base-TX (100 Mbps Ethernet IEEE 802.3u). The idea is that the "victim" and one of the firewall network interface cards (NICs) would be connected via one hub, while the "Linux attacker", "NT attacker" and other firewall NIC would be connected via the other hub as shown in the diagram below.

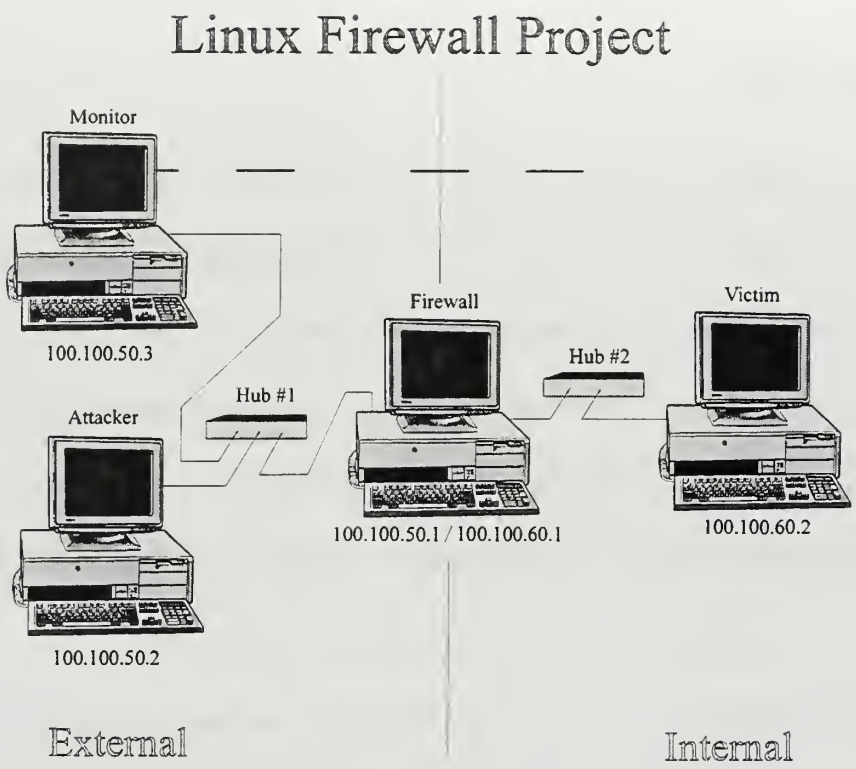

Figure 15. Ipchains Project System Configuration

While the hardware setup and configuration proved to be a fairly straightforward operation for the experienced system administrator, base knowledge in networking, routing and hardware/software configuration certainly have a bearing on system configuration. The first time system administrator, could take as long as a few days to complete this procedure.

The one snag that emerged came from the installed high performance Diamond Stealth III S500 Series video cards that turned out to be incompatible with the Linux Red Hat 6.0 software that was chosen as the test operating system. A list of compatible video

cards was provided with the documentation that came with the Redhat installation software. One of the recommended cards was procured and a hardware swap of the cards was performed, followed by a driver download. The chosen cards, ATI Graphics Accelerator cards, worked well with the exception of one of my machines. The problem manifested itself in the form of a group of vertical lines about one inch wide and the height of the screen that would appear whenever the computer was in the command line mode. After some experimentation, it was discovered that by changing the Xwindows size configuration, the "ghost" lines could be eliminated. Later versions of Redhat Linux, such as 6.2, have eliminated the video card compatibility problem.

B. SOFTWARE

The installation of the system software proved to be much more involved than expected. It turned out to be a trial and error operation. The situation involved configuring computers that were already booted in the Windows NT environment with the Linux operating system. Since all the computers were already loaded with Windows NT, all that was required was to load up two of the four computers with Linux, thereby making them "dual booting" machines, capable of working in either the NT or Linux environments. This, of course, is easier said than done. A series of drive reconfigurations and the handy use of the installation guide made the process somewhat less troublesome.

C. ROUTING

When configuring the firewall computer on the network, it is imperative that the system administrator enable routing traffic. The firewall computer must be configured as a router if two network interface cards will be employed. The following steps *must* be taken in order for your firewall to route traffic correctly:

1. Open linuxconf under the gnome/system menu
2. Select Client tasks
3. Select Routing and Gateways
4. Select Route to Other Networks
5. Click on the Add button
6. Enter the address of NIC #1 in the Gateway block
7. Enter the address of NIC #2 in the Destination block
8. Enter the same Netmask as used in the networks you previously configured
9. Click on the Accept button
10. Repeat the process to route traffic in the opposite direction
11. Click on the Add button
12. Enter the address of NIC #1 in the Gateway block
13. Enter the address of NIC #2 in the Destination block
14. Enter the same Netmask as used in the networks you previously configured
15. Click the Accept button
16. Finally, go back to Defaults under the Routing and gateways section of the gnome menu

17. Click "enable", then "accept" and begin routing traffic!

This is a crucial process that is not discussed in any of the Linux reference manuals I used in my research.

D. SECURITY ASSESSMENT WITHOUT FIREWALL

The first set of data recorded here is the result of a series of scans with the firewall completely disabled. After a quick review of the vulnerabilities revealed in this simple network application, the reasons for network security become very apparent. Nmap had no problem determining the list of hosts available on the network, the type of operating system running on the individual hosts, the vulnerable ports on those hosts and listing the port numbers and services available.

This first scan is a ping scan, a tool normally used by hackers to determine what hosts are available on the net in question. It uses ICMP echo requests and ACK techniques to accomplish this task.

[root@Attack /tools]# nmap –v –sP 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up.
Host (100.100.60.2) appears to be up.
Nmap run completed –2 IP addresses (2 hosts up) scanned in 171 seconds

The next scan was a FIN scan, which uses a bare (surprise) FIN packet as the probe to determine which ports are available on the host in question. It is a stealthy scan

and can be used in conjunction with the SYN scan to aid in determining operating system type.

[root@Attack /tools]# nmap –v –sF 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up…good.
Initiating FIN, NULL, UDP or Xmas stealth scan against (100.100.60.1)
The UDP or stealth FIN/NULL/XMAS scan took 10 seconds to scan 1511 ports.
Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
79	open	tcp	finger
98	open	tcp	linuxconf
111	open	tcp	sunrpc
113	open	tcp	auth
513	open	tcp	login
514	open	tcp	shell
515	open	tcp	printer
1067	open	tcp	instl_boots
6000	open	tcp	X11

Host (100.100.60.2) appears to be up…good.
Initiating FIN, NULL, UDP or Xmas stealth scan against (100.100.60.2)
The UDP or stealth FIN/NULL/XMAS scan took 1 seconds to scan 1511 ports.
No ports open for host (100.100.60.2)
Nmap run complete –2 IP addresses (2 hosts up) scanned in 181 seconds

The SYN scan illustrated below is referred to as a "half-open" stealth scan. It also searches for open ports on the desired hosts. The beauty of the SYN scan is that, as stated above, it can be used in conjunction with the FIN scan to help determine operating system. Since the FIN scan doesn't work against a Windows-based system, the scan will return that the host is up, but will show no ports open. However, when the SYN scan is performed on the same host, the system shows, not only up, but also what ports are

available. This combination, not to mention a review of those port that are up, will inevitably give away the type of operating system being used.

[root@Attack /tools]# nmap –v –sS 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up...good.
Initiating SYN half-open stealth scan against (100.100.60.1)
Adding TCP port 514 (state Open).
Adding TCP port 21 (state Open).
Adding TCP port 113 (state Open).
Adding TCP port 25 (state Open).
Adding TCP port 515 (state Open).
Adding TCP port 111 (state Open).
Adding TCP port 513 (state Open).
Adding TCP port 1067 (state Open).
Adding TCP port 79 (state Open).
Adding TCP port 6000 (state Open).
Adding TCP port 98 (state Open).
Adding TCP port 23 (state Open).
The SYNscan took 10 seconds to scan 1511 ports.
Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
79	open	tcp	finger
98	open	tcp	linuxconf
111	open	tcp	sunrpc
113	open	tcp	auth
513	open	tcp	login
514	open	tcp	shell
515	open	tcp	printer
1067	open	tcp	instl_boots
6000	open	tcp	X11

Host (100.100.60.1) appears to be up...good.
Initiating SYN half-open stealth scan against (100.100.60.2)
Adding TCP port 135 (state Open).
Adding TCP port 12345 (state Open).
Adding TCP port 12346 (state Open).
Adding TCP port 139 (state Open).

The SYN scan took 2 seconds to scan 1511 ports.
Interesting ports on (100.100.60.2):

Port	State	Protocol	Service
135	open	tcp	loc-srv
139	open	tcp	netbios-scan

Nmap run complete –2 IP addresses (2 hosts up) scanned in 181 seconds

The UDP scan below is used to determine which UDP (User Data Protocol, RFC 768) ports are open on the host. The technique sends zero byte udp packets to each port on the target machine. This is very useful in determining vulnerable UDP points of access.

[root@Attack /tools]# nmap –v –sU 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up…good.
Initiating FIN, NULL, UDP or Xmas stealth scan against (100.100.60.1)
Too many drops … increasing senddelay to 50000
Too many drops … increasing senddelay to 100000
Too many drops … increasing senddelay to 200000
Too many drops … increasing senddelay to 400000
Too many drops … increasing senddelay to 800000
The UDP or stealth FIN/NULL/XMAS scan took 1477 seconds to scan 1445 ports.
Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
111	open	tcp	sunrpc
517	open	tcp	shell
518	open	tcp	printer

Host (100.100.60.2) appears to be up…good.
Initiating FIN, NULL, UDP or Xmas stealth scan against (100.100.60.2)
The UDP or stealth FIN/NULL/XMAS scan took 5 seconds to scan 1445 ports.
Interesting ports on (100.100.60.2):

Port	State	Protocol	Service
135	open	udp	loc-srv
137	open	udp	netbios-ns
138	open	udp	netbios-dgm
161	open	udp	snmp
1025	open	udp	blackjack

Nmap run complete –2 IP addresses (2 hosts up) scanned in 1652 seconds

The RPC scan works in combination with the various port scan methods. It floods all the TCP/UDP ports found open with SunRPC program NULL commands in an attempt to determine whether they are RPC ports, and if so, what program and version number they are.

[root@Attack /tools]# nmap –v –sR 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
No tcp, udp, or ICMP scantype specified, assuming vanilla tcp connection ( ) scan.
Use –sP if you really don't want to portscan (and just want to see what hosts are up).

Host (100.100.60.1) appears to be up...good.
Initiating TCP connect ( ) scan against (100.100.60.1)
Adding TCP port 113 (state Open).
Adding TCP port 23 (state Open).
Adding TCP port 111 (state Open).
Adding TCP port 25 (state Open).
Adding TCP port 21 (state Open).
Adding TCP port 515 (state Open).
Adding TCP port 513 (state Open).
Adding TCP port 514 (state Open).
Adding TCP port 1067 (state Open).
Adding TCP port 79 (state Open).
Adding TCP port 6000 (state Open).
Adding TCP port 98 (state Open).
The TCP connect scan took 0 seconds to scan 1511 ports.
Initiating RPC scan against (100.100.60.1)
The RPC scan took 7 seconds to scan 1511 ports.
Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
79	open	tcp	finger
98	open	tcp	linuxconf
111	open	tcp	sunrpc
113	open	tcp	auth

513 open tcp login
514 open tcp shell
515 open tcp printer
1067 open tcp instl_boots
6000 open tcp X11

Host (100.100.60.2) appears to be up...good.
Initiating TCP connect ( ) scan against (100.100.60.2)
Adding TCP port 12346 (state Open).
Adding TCP port 139 (state Open).
Adding TCP port 135 (state Open).
Adding TCP port 12345 (state Open).

The TCP scan took 1 seconds to scan 1511 ports.
Initiating RPC scan against (100.100.60.2)
The RCP scan took 0 seconds to scan 1511 ports.
Interesting ports on (100.100.60.2):
Port State Protocol Service
135 open tcp loc-srv
139 open tcp netbios-ssn
12345 open tcp NetBus
12346 open tcp NetBus

Nmap run complete –2 IP addresses (2 hosts up) scanned in 178 seconds

E. BUILDING THE FIREWALL

Prior to running any further scans, the basic firewall was constructed. Initiation of the firewall was accomplished through a process of choosing the various services, then systematically appending the necessary rules to engage those options to the firewall chain.

However, before building the chain of filtering rules, it is first necessary to remove the possibility of any pre-existing firewall or filtering elements. This is

accomplished through the following command: (Note: A description of each rule will be preceded by a # symbol.)

#Flush any existing rules from all chains

ipchains -F

At this point, the system is in its default, accept everything, policy state. This is a good place to begin. The most logical way for building the firewall is to deny everything coming in and reject everything going out. Unless a rule is defined to explicitly allow a matching packet through, incoming packets are silently denied without notification to the remote sender, and outgoing packets are rejected and an ICMP error message is returned to the local sender. (Ziegler, 2000)

#Set the default policy to deny everything

ipchains –P input DENY
ipchains –P output REJECT
ipchains –P forward REJECT

From here, it is now possible to begin appending rules that allow specific traffic, actions or activities. Let's begin by enabling unrestricted traffic. This allows you to run any local network services you choose without having to worry about getting all the firewall rules specified.

#Unlimited traffic on the loopback interface

ipchains –A input –I $LOOPBACK_INTERFACE –j ACCEPT
ipchains –A output –I $LOOPBACK_INTERFACE –j ACCEPT

Now, it is necessary to establish some input chain filters based on source and destination addresses. These addresses will never be seen in legitimate incoming packets.

```
#Setting up IP spoofing protection
#Turn on Source Address Verification
for f in  /proc/sys/net/ipv4/conf/*/rp_filter; do
        echo 1 > $f
done

#Refuse spoofed packets claiming to be from you
ipchains –A input –i $EXTERNAL_INTERFACE –s $IPADDR –j DENY –l
```

The –l option enables logging for packets matching the rule. When a packet matches the rule, the event is logged in /var/log/messages.

The next several sets of rules refuse incoming and outgoing packets with any of the Class A, B or C private network addresses as their source or destination addresses. The remaining sets of rules deny malformed packets, Class D multicast addresses, Class E reserved net addresses, and outgoing multicast packets.

```
# Refuse packets claiming to be to or from a Class A private network
ipchains –A input –i EXTERNAL_INTERFACE –s $CLASS_A –j DENY
ipchains –A input –i EXTERNAL_INTERFACE –d $CLASS_A –j DENY
ipchains –A output –i EXTERNAL_INTERFACE –s $CLASS_A –j DENY -l
ipchains –A output –i EXTERNAL_INTERFACE –d $CLASS_A –j DENY -l

# Refuse packets claiming to be to or from a Class B private network
ipchains –A input –i EXTERNAL_INTERFACE –s $CLASS_B –j DENY
ipchains –A input –i EXTERNAL_INTERFACE –d $CLASS_B –j DENY
ipchains –A output –i EXTERNAL_INTERFACE –s $CLASS_B –j DENY -l
ipchains –A output –i EXTERNAL_INTERFACE –d $CLASS_B –j DENY -l

# Refuse packets claiming to be to or from a Class C private network
ipchains –A input –i EXTERNAL_INTERFACE –s $CLASS_C –j DENY
ipchains –A input –i EXTERNAL_INTERFACE –s $CLASS_C –j DENY
ipchains –A output –i EXTERNAL_INTERFACE –s $CLASS_C –j DENY -l
ipchains –A output –i EXTERNAL_INTERFACE –d $CLASS_C –j DENY -l
# Refuse packets claiming to be from the loopback interface
ipchains –A input –i EXTERNAL_INTERFACE –s $CLASS_A –j DENY
ipchains –A output –i EXTERNAL_INTERFACE –s $CLASS_A –j DENY -l
#Refuse malformed broadcast packets
ipchains –A input –i EXTERNAL_INTERFACE –s $BROADCAST_DEST –j DENY -l
```

```
ipchains –A input –i EXTERNAL_INTERFACE –d $BROADCAST_DEST –j DENY –l

# Refuse Class D multicast addresses illegal only as a source address.
ipchains –A input –i EXTERNAL_INTERFACE –s $CLASS_D_MULTICAST –j
DENY -l
ipchains –A input –i EXTERNAL_INTERFACE –s $CLASS_D_MULTICAST –j
DENY -l

# Refuse Class E reserved IP addresses
ipchains –A input –i EXTERNAL_INTERFACE –s $CLASS_E_RESERVED_NET –j
DENY –l
```

The next several rules apply to error and status control messages. Four ICMP control and status messages need to pass through the firewall: Source Quench, Parameter Problem, incoming Destination Unreachable and outgoing Destination Unreachable, subtype Fragmentation Needed. Four other ICMP message types are optional: Echo Request, Echo Reply, other outgoing Destination Unreachable subtypes and Time Exceeded.

```
#Allows source Quench Control (Type 4) Messages
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –s $ANYWHERE 4 –d
$IPADDR –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p icmp –s $ IPADDR 4 –d
$ANYWHERE –j ACCEPT

#Allows parameter Problem Status (Type 12) Messages
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –s $ANYWHERE 12 –d
$IPADDR –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p icmp –s $ IPADDR 12 –d
$ANYWHERE –j ACCEPT


#Allows destination Unreachable Error (Type 3) Messages
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –s $ANYWHERE 3 –d
$IPADDR –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p icmp –s $ IPADDR 3 –d
$ANYWHERE –j ACCEPT
```

```
#Allows time Exceeded Status (Type 11) Messages
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –s $ANYWHERE 11 –d
$IPADDR –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p icmp –s $ IPADDR 11 –d
$ANYWHERE –j ACCEPT

#Allows outgoing ping to Remote Hosts anywhere
ipchains –A output –i EXTERNAL_INTERFACE –p icmp –s $ IPADDR 8 –d
$ANYWHERE –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –s $ANYWHERE 8 –d
$IPADDR –j ACCEPT

#Allows incoming ping from trusted hosts
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –s $ANYWHERE 8 –d
$IPADDR –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p icmp –s $ IPADDR 8 –d
$ANYWHERE –j ACCEPT

#Blocking Incoming and Outgoing smurf Attacks
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –d $BROADCAST_DEST –j
DENY -l
ipchains –A output –i EXTERNAL_INTERFACE –p icmp -d $BROADCAST_DEST –j
REJECT –l
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –d $NETMASK –j DENY
ipchains –A output –i EXTERNAL_INTERFACE –p icmp -d $NETMASK –j REJECT
ipchains –A input –i EXTERNAL_INTERFACE –p icmp –d $NETWORK –j DENY
ipchains –A output –i EXTERNAL_INTERFACE –p icmp -d $NETWORK –j REJECT
```

LAN services often run on unprivileged ports. For TCP-based services, a connection attempt to one of these services can be distinguished from an outgoing connection with a client using one of these unprivileged ports through the state of the SYN and ACK bits. Incoming connection attempts to these ports should be blocked for security purposes. Outgoing connections should be blocked for protection against mistakes on the internal end and to log potential internal security problems. The first rule instituted here blocks local clients from initiating a connection request to a remote Open

Window manager on port 2000. Next come rules denying X Window connections on port 6000 and SOCKS server connections on port 1080.

```
#Open Windows: establishing a connection
ipchains –A output –i EXTERNAL_INTERFACE –p tcp –y -s $IPADDR –d
$ANYWHERE $OPENWINDOWS_PORT –j REJECT

#X Windows: establishing a remote connection
ipchains –A output –i EXTERNAL_INTERFACE –p tcp –y -s $IPADDR –d
$ANYWHERE $XWINDOW_PORT –j REJECT

#X Windows: incoming connection attempt
ipchains –A output –i EXTERNAL_INTERFACE –p tcp –y -d $IPADDR –d $IPADDR
$XWINDOW_PORT –j REJECT
```

As a datagram service, UDP doesn't have a connection state associated with it. Access to UDP services should simply be blocked. Explicit exceptions are made to accommodate DNS and any of the few other UDP-based Internet services. Fortunately, the common UDP Internet services are often the types that are used between a client and a specific server. The filtering rules can often allow exchanges with one specific remote host. NFS is the main UDP service to be concerned with. NFS runs on unprivileged port 2049. Unlike the previous TCP-based services, NFS is primarily a UDP-based service.

```
#NFS: UDP connections (port 2049)
ipchains –A input –i EXTERNAL_INTERFACE –p udp –d $IPADDR $NFS_PORT –j
DENY -l


#NFS: TCP connections (port 2049)
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -y –d $IPADDR $NFS_PORT
–j DENY -l
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -y –d $IPADDR $NFS_PORT
–j DENY –l
```

In order enable basic services, rules must be appended that allow for the domain name server (DNS) that translates between hostnames and their associated IP addresses and IDENT, which provides the username or ID associated with a connection. IDENT is commonly requested by a remote mail server when you send mail.

```
#Allowing DNS lookups as a client
ipchains –A output –i EXTERNAL_INTERFACE –p udp -s $IPADDR
$UNPRIVPORTS –d $NAMESERVER 53 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p udp -s  $NAMESERVER 53 –d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Allowing DNS lookups as a peer-to-peer, forwarding server
ipchains –A output –i EXTERNAL_INTERFACE –p udp -s $IPADDR 53 –d
$NAMESERVER 53 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p udp –s $NAMESERVER 53 –d
$IPADDR 53 –j ACCEPT

#Allowing outgoing AUTH requests as a client
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d $ANYWHERE 113 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y -s $ANYWHERE 113 –d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Filtering incoming AUTH requests to your server
ipchains –A input –i EXTERNAL_INTERFACE –p tcp -s $ANYWHERE
$UNPRIVPORTS –d $IPADDR 113 –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y -s $IPADDR 113 –d
$ANYWHERE $UNPRIVPORTS –j ACCEPT
```

This section is dedicated to the enabling of some common TCP services such as email, usenet, telnet, ssh, ftp, finger, whois and gopher. Email is probably the most

common TCP service requested. As such, the following rules are associated with the email ports, 25, 110 and 143.

```
#Relaying outgoing mail through an external ISP gateway SMTP server
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
-d $SMTP_GATEWAY 25 –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y –s $SMTP_GATEWAY 25
–d $IPADDR $UNPRIVPORTS –j ACCEPT

#Sending mail to any external mail server
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
-d $ANYWHERE 25 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 25 –d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Receiving mail as a local SMTP server (TCP port 25)
ipchains –A input –i EXTERNAL_INTERFACE –p tcp -s $ANYWHERE
$UNPRIVPORTS –d  $IPADDR 25 –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y –s $IPADDR 25 –d
$ANYWHERE $UNPRIVPORTS –j ACCEPT

#Retrieving mail as a POP client (TCP port 110)
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $POP_SERVER 110 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $POP_SERVER 110 –d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Receiving mail as a IMAP client (TCP port 143)
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $IMAP_SERVER 143 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $IMAP_SERVER 143 –d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Sending mail as an SMTP client and receiving mail as a POP client
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $SMTP_GATEWAY 25 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $SMTP_GATEWAY 25
–d $IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $POP_SERVER 110 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $POP_SERVER 110 –d
$IPADDR $UNPRIVPORTS –j ACCEPT
```

```
#Sending mail as an SMTP client and receiving mail as an IMAP client
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $SMTP_GATEWAY 25 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $SMTP_GATEWAY 25
–d $IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $IMAP_SERVER 143 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $IMAP_SERVER 143 –d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Sending mail as an SMTP client and receiving mail as an SMTP server
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $SMTP_GATEWAY 25 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $SMTP_GATEWAY 25
–d $IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp -s $ANYWHERE
$UNPRIVPORTS –d  $IPADDR 25 –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y –s $IPADDR 25 –d
$ANYWHERE $UNPRIVPORTS –j ACCEPT

#Sending mail as an SMTP server and receiving mail as an SMTP server
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $ANYWHERE 25 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 25 –d
$IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp -s $ANYWHERE
$UNPRIVPORTS –d  $IPADDR 25 –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y –s $IPADDR 25 –d
$ANYWHERE $UNPRIVPORTS –j ACCEPT

#Reading and posting news as a Usenet client
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $NEWS_SERVER 119 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $NEWS_SERVER 119 -d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Allowing incoming access to the local server
ipchains –A input –i EXTERNAL_INTERFACE –p tcp -s $ANYWHERE
$UNPRIVPORTS –d  $IPADDR 23 –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y –s $IPADDR 23 –d
$ANYWHERE $UNPRIVPORTS –j ACCEPT
```

```
#Allowing client access to remote SSH servers
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $ANYWHERE 22 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 22 –d
$IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $SSH_PORTS –d
$ANYWHERE 22 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 22 –d
$IPADDR $SSH_PORTS –j ACCEPT

#Allowing remote client access to your local SSH server
ipchains –A input –i EXTERNAL_INTERFACE –p tcp -s $ANYWHERE
$UNPRIVPORTS –d  $IPADDR 22 –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y –s $IPADDR 22 –d
$ANYWHERE $UNPRIVPORTS –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp -s $ANYWHERE
$SSH_PORTS –d  $IPADDR 22 –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y –s $IPADDR 22 –d
$ANYWHERE $SSH_PORTS –j ACCEPT

#Allowing outgoing FTP requests
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $ANYWHERE 21 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 21 –d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Allowing normal port mode FTP data channels
ipchains –A input –i EXTERNAL_INTERFACE –p tcp -s $ANYWHERE 20 –d
$IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A output –i EXTERNAL_INTERFACE –p tcp ! -y –s $IPADDR
$UNPRIVPORTS –d $ANYWHERE 20 –j ACCEPT

#Accessing remote web sites as a client
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $ANYWHERE 80 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 80 –d
$IPADDR $UNPRIVPORTS –j ACCEPT

#Accessing remote finger servers as a client
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS
–d  $ANYWHERE 79 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 79 –d
$IPADDR $UNPRIVPORTS –j ACCEPT
```

#Allowing whois queries to an official remote server
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS –d $ANYWHERE 43 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 43 –d $IPADDR $UNPRIVPORTS –j ACCEPT

#Allowing gopher connection to a remote server
ipchains –A output –i EXTERNAL_INTERFACE –p tcp -s $IPADDR $UNPRIVPORTS –d $ANYWHERE 70 –j ACCEPT
ipchains –A input –i EXTERNAL_INTERFACE –p tcp ! -y –s $ANYWHERE 70 –d $IPADDR $UNPRIVPORTS –j ACCEPT

This concludes the installation of the firewall rules utilized in this experiment. The next step will be to conduct full scale vulnerability testing using the Nmap scanning software.

F. SECURITY ASSESSMENT WITH FIREWALL

This section makes it apparent why the firewall is imperative. As recommended, all access to and from the internal network was completely terminated. Only after verification that all access had actually been shut down, were various packet filtering exceptions allowing various services activated. As can be seen, in spite of various attempts at access, penetration of the firewall was futile. Access to the firewall router network interface card, 100.100.60.1, was continued, while access to the internal network was prohibited.

[root@Attack /tools]# nmap –v –sP 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up.
Host (100.100.60.2) appears to be down.
Nmap run completed –2 IP addresses (1 host up) scanned in 171 seconds

[root@Attack /tools]# nmap –v –sF 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up…good.
Initiating FIN, NULL, UDP or Xmas stealth scan against (100.100.60.1)
The UDP or stealth FIN/NULL/XMAS scan took 9 seconds to scan 1511 ports.
Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
79	open	tcp	finger
98	open	tcp	linuxconf
111	open	tcp	sunrpc
113	open	tcp	auth
513	open	tcp	login
514	open	tcp	shell
515	open	tcp	printer
1067	open	tcp	instl_boots
6000	open	tcp	X11

Host (100.100.60.2) appears to be down, skipping it.
Nmap run complete –2 IP addresses (1 host up) scanned in 94 seconds

[root@Attack /tools]# nmap –v –sS 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up…good.
Initiating SYN half-open stealth scan against (100.100.60.1)
Adding TCP port 514 (state Open).
Adding TCP port 21 (state Open).
Adding TCP port 113 (state Open).
Adding TCP port 25 (state Open).
Adding TCP port 515 (state Open).
Adding TCP port 111 (state Open).
Adding TCP port 513 (state Open).
Adding TCP port 1084 (state Open).
Adding TCP port 79 (state Open).
Adding TCP port 6000 (state Open).
Adding TCP port 98 (state Open).
Adding TCP port 23 (state Open).
The SYNscan took 10 seconds to scan 1511 ports.

Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
79	open	tcp	finger
98	open	tcp	linuxconf
111	open	tcp	sunrpc
113	open	tcp	auth
513	open	tcp	login
514	open	tcp	shell
515	open	tcp	printer
1084	open	tcp	ansoft-lm-2
6000	open	tcp	X11

Host (100.100.60.1) appears to be down, skipping it.
Nmap run complete –2 IP addresses (1 host up) scanned in 90 seconds

[root@Attack /tools]# nmap –v –sU 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up…good.
Initiating FIN, NULL, UDP or Xmas stealth scan against (100.100.60.1)
Too many drops … increasing senddelay to 50000
Too many drops … increasing senddelay to 100000
Too many drops … increasing senddelay to 200000
Too many drops … increasing senddelay to 400000
Too many drops … increasing senddelay to 800000
The UDP or stealth FIN/NULL/XMAS scan took 1474 seconds to scan 1445 ports.
Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
111	open	tcp	sunrpc
517	open	tcp	talk
518	open	tcp	ntalk

Host (100.100.60.2) appears to be down, skipping it.
Nmap run complete –2 IP addresses (1 host up) scanned in 1560 seconds

[root@Attack /tools]# nmap –v –sR 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
No tcp, udp, or ICMP scantype specified, assuming vanilla tcp connection ( ) scan.
Use –sP if you really don't want to portscan (and just want to see what hosts are up).

Host (100.100.60.1) appears to be up...good.
Initiating TCP connect ( ) scan against (100.100.60.1)
Adding TCP port 113 (state Open).
Adding TCP port 23 (state Open).
Adding TCP port 111 (state Open).
Adding TCP port 25 (state Open).
Adding TCP port 21 (state Open).
Adding TCP port 515 (state Open).
Adding TCP port 513 (state Open).
Adding TCP port 514 (state Open).
Adding TCP port 1084 (state Open).
Adding TCP port 79 (state Open).
Adding TCP port 6000 (state Open).
Adding TCP port 98 (state Open).
The TCP connect scan took 0 seconds to scan 1511 ports.
Initiating RPC scan against (100.100.60.1)
The RPC scan took 6 seconds to scan 1511 ports.
Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
79	open	tcp	finger
98	open	tcp	linuxconf
111	open	tcp	sunrpc (portmapper V2)
113	open	tcp	auth
513	open	tcp	login
514	open	tcp	shell
515	open	tcp	printer
1084	open	tcp	ansoft-ln-2
6000	open	tcp	X11

Host (100.100.60.2) appears to be down, skipping it.
Nmap run complete –2 IP addresses (1 host up) scanned in 92 seconds

[root@Attack /tools]# nmap –v –sS –O 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up...good.
Initiating SYN half-open stealth scan against (100.100.60.1)
Adding TCP port 514 (state Open).
Adding TCP port 21 (state Open).

Adding TCP port 113 (state Open).
Adding TCP port 25 (state Open).
Adding TCP port 515 (state Open).
Adding TCP port 111 (state Open).
Adding TCP port 513 (state Open).
Adding TCP port 1084 (state Open).
Adding TCP port 79 (state Open).
Adding TCP port 6000 (state Open).
Adding TCP port 98 (state Open).
Adding TCP port 23 (state Open).
The SYNscan took 4 seconds to scan 1511 ports.
For OSScan assuming that port 21 is open and port 876 is closed and neither are firewalled
Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
79	open	tcp	finger
98	open	tcp	linuxconf
111	open	tcp	sunrpc
113	open	tcp	auth
513	open	tcp	login
514	open	tcp	shell
515	open	tcp	printer
6000	open	tcp	X11

TCP Sequence Prediction: Class=random positive increments
Difficulty=1661750 (Good Luck!)

Sequence numbers: 1C754EB1 1C847202 1C8C57E8 1C528E49 1C97EF4E 1CA6D519
Remote operating system guess: Linux 2.1.122-2.2.13

Host (100.100.60.1) appears to be down, skipping it.
Nmap run complete –2 IP addresses (1 host up) scanned in 89 seconds

[root@Attack /tools]# nmap –v –sX –p 22,53,110,111,143,2752,4564,6000 100.100.60.1-2

Starting nmap V. 2.3BETA14 by fyodor@insecure .org(www.insecure.org/nmap/)
Host (100.100.60.1) appears to be up…good.
Initiating FIN, NULL, UDP or Xmas stealth scan against (100.100.60.1)
The UDP or stealth FIN/NULL/XMAS scan took 1 seconds to scan 8 ports.

Interesting ports on (100.100.60.1):

Port	State	Protocol	Service
111	open	tcp	sunrpc
6000	open	tcp	X11

Host (100.100.60.2) appears to be down, skipping it.
Nmap run complete –2 IP addresses (1 host up) scanned in 87 seconds

THIS PAGE INTENTIONALLY LEFT BLANK

IX. CONCLUSIONS

A. SUMMARY OF FINDINGS

The conclusion of this research has led to a few basic deductions. The first and foremost is that firewalls, in general, are a step in the right direction to enhanced network security, but they are far from a comprehensive solution to protecting one's information assets. Network users and system administrators are still required to operate under, maintain, and implement the strictest of security measures needed to ensure a system is safe. Usually it is the human aspect that fails the firewalls rather than the firewall failing the network. Improper installation and implementation of a firewall are the biggest security threat.

Linux proved to be more difficult to master than the Windows 2000/NT environments. For those with little or no UNIX background, the learning curve here is steep. However, once a basic understanding of the operating system is attained, mastery of Linux system administration is manageable. The Linux firewall, Ipchains was also relatively easy to understand and maintain.

Ipchains proved to be significantly more robust than its Microsoft-based counterparts, probably owing, in large part, to the nature of its open source code. New information on current releases, system vulnerabilities and software patches are released practically daily. While, the Windows system administrator must rely on the timeline of Microsoft (and other closed source firewall systems) to fix vulnerabilities and release

updates to the current software package in a timely manner. One can instantly see the advantage of being able to make software corrections in a real time manner, vice waiting for the next release.

As far as advice for the system administrator using Linux as an operating system and Ipchains as the firewall, the key will be to continue to expand a basic body of knowledge of Linux systems. Keep pace with the hacker's world and stay plugged in to emerging trends in security vulnerabilities and software patches. This, in conjunction with other security enhancements, will prove to be as robust a firewall security system as one could expect to get, in spite of its bargain basement pricing with respect to commercial firewall developers.

APPENDIX A: IPCHAINS RULES AND ACTIONS

1. IPCHAINS ACTIONS

Parameter	Action
-A	Appends one or more rules to the end of a selected chain. If the rule applies to more than one set of IP address and port number combination, the rule will be added for each combination.
-D	Deletes one or more rules from the selected chain. You can either specify a rule number or specify the parameters for which a rule should be deleted. (For instance, you can delete all rules that apply to IP address 192.168.2.19.)
-R	Replaces a rule in the selected chain. If the source and/or destination names match more than one entry, the replacement will fail, and the program will return an error.
-I	Inserts one or more rules in the selected chain at a given rule number.
-L	Lists a chain. If no specific chain is requested, all of the chains are listed.
-F	Flushes the selected chain. This is the same as deleting all of the entries in a particular chain.
-Z	Zeros out all the counters that ipchains keeps for accounting purposes.
-N	Creates a new chain.
-X	Deletes a chain. A chain must be empty before you can do this – that is, you need to flush the chain first.
-P	Sets the default policy for a given chain.

2. IPCHAINS PARAMETER TYPES

Type of Parameter	Meaning
Chain	The name of the chain that the invocation of ipchains is going to work on. The three default chains are input, forward, output.
Rulespec	The rule specification.
Rulenum	The number of the rule you want to work on.
Target	The action the kernel should take when it finds a packet that matches a rule listed in the chain. Valid targets are ACCEPT, REJECT, and DENY. ACCEPT means that the packet is allowed to pass through whatever chain is currently being processed. DENY means the packet is dropped and the system simply pretends as if it never got it. REJECT is the friendly version of DENY: It refuses to accept the packet, but it sends an ICMP message to the sender indicating so.
Options	Additional modifiers that can be applied to each rule. These are, of course, not required.

3. OPTION CHOICES

Option	Description
-b	Applies the rule bidirectionally. This is effectively the same as setting up an additional rule where the source and destination information are reversed.
-v	Verbose output (provides greater detail)
-n	Gives numeric output. Normally, listing all the rules or changing the rules with the verbose option will result in all of the IP addresses getting their names resolved so that the

output will be more readable. If you are debugging a problem and host name resolution is not available, you should use this option so the system doesn't stall while waiting for DNS requests to time out.

-l — Turns on kernel logging for all packets that match the rules.

-x — Shows exact numbers when printing rule information with the –L action. Typically, output is rounded to the nearest 1,000 or 1,000,000.

-y — Matches only packets with the SYN bit set, but not the ACK or FIN bits set. The SYN bit in a TCP header is used to initiate connections. Typically, this option is used to block incoming connections but allow outgoing connections. You can prefix this option with an exclamation mark (!) to invert it, thereby matching all packets where the SYN bit isn't set and the ACK or FIN bits are set.

4. IPCHAINS RULE SPECIFICATIONS

-p *protocol* — Specifies the type of IP packet to which this rule should apply. Valid protocols are those that are listed in the /etc/protocols file or a numeric value for any protocol that isn't listed there. Typical values are tcp, udp, icmp, or all, where all refers to all of the valid protocols. If you prefix the protocol with an exclamation point (!), you invert the test. For example saying "-p !icmp" effectively means "for all protocols that are not icmp."

-s *source/mask port* — States what the source IP address must be for this rule to match. Typical uses are to deny networks that are known to be hostile or untrustworthy. In this option, *source* is the IP address and *mask* is the netmask that can be represented either by using dotted decimal notation (such as 255.255.255.0) or by specifying the number of bits in the netmask (for example, 25 is equal to a netmask of 255.255.255.128). The *port* value allows you to specify the source port number on which you want to act. You can use a range of values by using the format *port1:port2*, where

	port1 is the lower bound and port2 is the upper bound. Port numbers are optional. For example, to specify all ports in the range 1024 to 65535 from any host in the 192.168.42.0 network, we would say "-s 192.168.42.024 1024:65535." Prefixing either an IP/netmask combination or port number with an exclamation point (!) negates the expression. For example, to specify any host except 192.168.42.7, we would say "-s!192.168.42.7."
--source-port port	Specifies a port (or port range if *port* is in the format of *port1:port2*, where port 1 is the lowest port number, and port2 is the highest port number) without having to specify an IP address. If you want to set up a rule for all hosts whose source port is 0 to 1023, for example, you would use "--source-port 0:1023."
-d destination/mask port	The same as the -s option, except this option applies to the destination instead of the source IP address.
--destination-port port	The same as the –source/port option, except this option applies to the destination port rather than the source port.
--icmp-type typename	Applies a rule to a ICMP packet whose type is *typename*. You can run "ipchains –h icmp" to see a list of types. The most common type is the echo-reply, which is used for the ping program. Prefixing *typename* with an exclamation point (!) inverts the rule. For example, to set up a rule for all ICMP packets except port-unreachable messages, we would say "—icmp-type ! port-unreachable."
-j target	Specifies that the action to perform if a rule matches should be a *target*, where *target* is either a user-defined chain or one of the predefined targets (ACCEPT, DENY, REJECT).
-i interface	Specifies the interface to which this rule should apply, where *interface* is the name of the device (such as eth0). If this option is not specified for a rule, it is assumed that the rule will apply to all interfaces. If *interface* is prefixed with an exclamation point (!), it inverts the rule. For example, to apply a rule to all interfaces except ppp1, we would say "-I ppp1."

-f	Applies this rule to any packet fragment. Because the header information in an IP fragment is not included with each fragment, it is impossible to apply any existing rules to it. You can avoid this mess altogether by configuring the kernel to defragment all packets before delivering them. If you do want to allow fragments from certain interfaces, use this in conjunction with the –i option. By prefixing the –f with an exclamation point (!), you can invert the meaning. For example, to say that we want to reject all fragments except those coming from eth0, we would use "! –f –I eth0 –j ALLOW." (Ziegler, 2000)

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: NMAP

Nmap is designed to allow system administrators and curious individuals to scan large networks to determine which hosts are up and what services they are offering. Nmap supports a large number of scanning techniques such as UDP, TCP connect (), TCP SYN (half open), ftp proxy (bounce attack), Reverse-indent, ICMP (ping sweep), FIN, ACK sweep, Xmas Tree, SYN sweep, and Null scan. Nmap also offers a number of advanced features such as remote OS detection via TCP/IP fingerprinting, stealth scanning, dynamic delay and retransmission calculations, parallel scanning, detection of down hosts via parallel pings, decoy scanning, port filtering detection, direct (non-portmapper) RPC scanning, fragmentation scanning, and flexible target and port specification.

Significant effort has been put into decent nmap performance for non-root users. Unfortunately, many critical kernel interfaces (such as raw sockets) require root privileges. Nmap should be run as root whenever possible.

The result of running Nmap is usually a list of interesting ports on the machine(s) being scanned (if any). Nmap always gives the port's "well known" service name (if any), number, state, and protocol. The state is either 'open', 'filtered', or 'unfiltered'. Open means that the target machine will accept ( ) connections on that port. Filtered means that a firewall, filter, or other network obstacle is covering the port and preventing Nmap from determining whether the port is open. Unfiltered means that the port is known by Nmap to be closed and no firewall/filter seems to be interfering with Nmap's

attempts to determine this. Unfiltered ports are the common case and are only shown when most of the scanned ports are in the filtered state.

Depending on options used, Nmap may also report the following characteristics of the remote host: OS in use, TCP sequencability, usernames running the programs which have bound to each port, the DNS name, whether the host is a Smurf address, and a few others.

Options that make sense together can generally be combined. Some options are specific to certain scan modes. Nmap tries to catch and warn the user about inappropriate or unsupported option combinations.

SCAN TYPES

-sT TCP connect() scan: This is the most basic form of TCP scanning. The connect() system call provided by your operating system is used to open a connection to every interesting port on the machine. If the port is listening, connect() will succeed, otherwise the port isn't reachable. One strong advantage to this technique is that you don't need any special privileges. Any user on most UNIX boxes is free to use this call. This sort of scan is easily detectable as target host logs will show a bunch of connection and error messages for the services which accept() the connection just to have it immediately shutdown.

-sS TCP SYN scan: This technique is often referred to as "half-open" scanning, because you don't open a full TCP connection. You send a SYN packet, as if you are going to open a real connection and you wait for a response. A SYN|ACK indicates

the port is listening. A RST is indicative of a non-listener. If a SYN|ACK is received, a RST is immediately sent to tear down the connection (actually our OS kernel does this for us). The primary advantage to this scanning technique is that fewer sites will log it. Unfortunately you need root privileges to build these custom SYN packets.

-sF -sX –sN

Stealth FIN, Xmas Tree, or Null scan modes: There are times when even SYN scanning isn't clandestine enough. Some firewalls and packet filters watch for SYNs to restricted ports, and programs like Synlogger and Courtney are available to detect these scans. These advanced scans, on the other hand, may be able to pass through unmolested. The idea is that closed ports are required to reply to your probe packet with an RST, while open ports must ignore the packets in question (see RFC 793 pp 64). The FIN scan uses a bare (surprise) FIN packet as the probe, while the Xmas tree scan turns on the FIN, URG, and PUSH flags. The Null scan turns off all flags. Unfortunately Microsoft (like usual) decided to completely ignore the standard and do things their own way. Thus this scan type will not work against systems running Windows95/NT. On the positive side, this is a good way to distinguish between the two platforms. If the scan finds open ports, you know the machine is not a Windows box. If a -sF,-sX, or -sN scan shows all ports closed, yet a SYN (-sS) scan shows ports being opened, you are probably looking at a Windows box. This is less useful now that nmap has proper OS detection built in. There are also a few other systems that are broken in the same

way Windows is. They include Cisco, BSDI, HP/UX, MVS, and IRIX. All of the above send resets from the open ports when they should just drop the packet.

-sP Ping scanning: Sometimes you only want to know which hosts on a network are up. Nmap can do this by sending ICMP echo request packets to every IP address on the networks you specify. Hosts that respond are up. Unfortunately, some sites such as microsoft.com block echo request packets. Thus nmap can also send a TCP ack packet to (by default) port 80. If we get an RST back, that machine is up. A third technique involves sending a SYN packet and waiting for a RST or a SYN/ACK. For non-root users, a connect() method is used. By default (for root users), nmap uses both the ICMP and ACK techniques in parallel. You can change the **-P** option described later. Note that pinging is done by default anyway, and only hosts that respond are scanned. Only use this option if you wish to ping sweep **without** doing any actual port scans.

-sU UDP scans: This method is used to determine which UDP (User Datagram Protocol, RFC 768) ports are open on a host. The technique is to send 0 byte udp packets to each port on the target machine. If we receive an ICMP port unreachable message, then the port is closed. Otherwise we assume it is open. Some people think UDP scanning is pointless. I usually remind them of the recent Solaris rcpbind hole. Rpcbind can be found hiding on an undocumented UDP port somewhere above 32770. So it doesn't matter that 111 is blocked by the firewall. But can you find which of the more than 30,000 high ports it is listening on? With a UDP scanner you

can! There is also the cDc Back Orifice backdoor program, which hides on a configurable UDP port on Windows machines. Not to mention the many commonly vulnerable services that utilize UDP such as snmp, tftp, NFS, etc. Unfortunately UDP scanning is sometimes painfully slow since most hosts implement a suggestion in RFC 1812 (section 4.3.2.8) of limiting the ICMP error message rate. For example, the Linux kernel (innet/ipv4/icmp.h) limits destination unreachable message generation to 80 per 4 seconds, with a 1/4 second penalty if that is exceeded. Solaris has much more strict limits (about 2 messages per second) and thus takes even longer to scan. *nmap* detects this rate limiting and slows down accordingly, rather than flood the network with useless packets that will be ignored by the target machine. As is typical, Microsoft ignored the suggestion of the RFC and does not seem to do any rate limiting at all on Win95 and NT machines. Thus we can scan all 65K ports of a Windows machine **very** quickly.

-sA ACK scan: This advanced method is usually used to map out firewall rule sets. In particular, it can help determine whether a firewall is stateful or just a simple packet filter that blocks incoming SYN packets. This scan type sends an ACK packet (with random looking acknowledgement/sequence numbers) to the ports specified. If a RST comes back, the ports is classified as "unfiltered". If nothing comes back (or if an ICMP unreachable is returned), the port is classified as "filtered". Note that *nmap* usually doesn't print "unfiltered" ports, so getting **no** ports shown in the

output is usually a sign that all the probes got through (and returned RSTs). This scan will obviously ever show ports in the "open" state.

-sW Window scan: This advanced scan is very similar to the ACK scan, except that it can sometimes detect open ports as well as filtered/nonfiltered due to anomaly in the TCP window size reporting by some operating systems. Systems vulnerable to this include at least some versions of AIX, Amiga, BeOS, BSDI, Cray, Tru64 UNIX, DG/UX, OpenVMS, Digital UNIX, FreeBSD, HP-UX, OS/2, IRIX, MacOS, NetBSD, OpenBSD, OpenStep, QNX, Rhapsody, SunOS 4.X, Ultrix, VAX, and VxWorks. See the nmap-hackers mailing list archive for a full list.

-sR RPC scan. This method works in combination with the various port scan methods of Nmap. It takes all the TCP/UDP ports found open and then floods them with SunRPC program NULL commands in an attempt to determine whether they are RPC ports, and if so, what program and version number they serve up. Thus you can effectively obtain the same info as firewall (or protected by TCP wrappers). Decoys do not currently work with RPC scan, at some point I may add decoy support for UDP RPC scans.

-b <ftp relay host> FTP bounce attack: An interesting "feature" of the ftp protocol (RFC 959) is support for "proxy" ftp connections. In other words, I should be able to connect from evil.com to the FTP server of target.com and request that the server end a file ANYWHERE on the internet! Now this may have worked well in 1985 when the RFC was written. But in today's Internet, we can't have people hijacking ftp servers

and requesting that data be spit out to arbitrary points on the internet. As *Hobbit* wrote back in 1995, this protocol flaw "can be used to post virtually untraceable mail and news, hammer on servers at various sites, fill up disks, try to hop firewalls, and generally be annoying and hard to track down at the same time." What we will exploit this for is to scan TCP ports from a "proxy" ftp server. Thus you could connect to an ftp server behind a firewall, and then scan ports that are more likely to be blocked (139 is a good one). If the ftp server allows reading from and writing to some directory (such as /incoming), you can send arbitrary data to ports that you do find open (nmap doesn't do this for you though). The argument passed to the 'b' option is the host you want to use as a proxy, in standard URL notation. The format is: *username:password@server:port*. Everything but *server* is optional.

GENERAL OPTIONS

None of these options are required but some can be quite useful.

-P0 Do not try and ping hosts at all before scanning them. This allows the scanning of networks that don't allow ICMP echo requests (or responses) through their firewall. microsoft.com is an example of such a network, and thus you should always use **-P0** or **-PT80** when portscanning microsoft.com.

-PT Use TCP "ping" to determine what hosts are up. Instead of sending ICMP echo request packets and waiting for a response, we spew out TCP ACK packets throughout the target network (or to a single machine) and then wait for responses to trickle back. Hosts that are up should respond with a RST. This option preserves the

efficiency of only scanning hosts that are up while still allowing you to scan networks/hosts that block ping packets. For non root users, we use connect(). To set the destination port of the probe packets use -PT<port number>. The default port is 80, since this port is often not filtered out.

-PS This option uses SYN (connection request) packets instead of ACK packets for root users. Hosts that are up should respond with a RST (or, rarely, a SYN|ACK).

-PI This option uses a true ping (ICMP echo request) packet. It finds hosts that are up and also looks for subnet-directed broadcast addresses on your network. These are IP addresses, which are externally reachable and translate to a broadcast of incoming IP packets to a subnet of computers. These should be eliminated if found as they allow for numerous denial of service attacks (Smurf is the most common).

-PB This is the default ping type. It uses both the ACK (**-PT**) and ICMP (**-IP**) sweeps in parallel. This way you can get firewalls that filter either one (but not both).

-O This option activates remote host identification via TCP/IP fingerprinting. In other words, it uses a bunch of techniques to detect subtleties in the underlying operating system network stack of the computers you are scanning. It uses this information to create a 'fingerprint' which it compares with its database of known OS fingerprints (the nmap-os-fingerprints file) to decide what type of system you are scanning. If you can't send the IP address, the next best thing is to run nmap with the **-d** option and send me the three fingerprints that should result along with the OS name

and version number. By doing this you contribute to the pool of operating systems known to nmap and thus it will be more accurate for everyone.

-I This turns on TCP reverse ident scanning. As noted by Dave Goldsmith in a 1996 Bugtraq post, the ident protocol (rfc 1413) allows for the disclosure of the username that owns any process connected via TCP, even if that process didn't initiate the connection. So, you can, for example, connect to the http port and then use identd to find out whether the server is running as root. This can only be done with a full TCP connection to the target port (i.e. the -sT scanning option). When **-I** is used, the remote host's identd is queried for each open port found. Obviously this won't work if the host is not running identd.

-f This option causes the requested SYN, FIN, XMAS, or NULL scan to use tiny fragmented IP packets. The idea is to split up the TCP header over several packets to make it harder for packet filters, intrusion detection systems, and other annoyances to detect what you are doing. Be careful with this! Some programs have trouble handling these tiny packets. My favorite sniffer segmentation faulted immediately upon receiving the first 36-byte fragment. After that comes a 24 byte one! While this method won't get by packet filters and firewalls that queue all IP fragments (like the CONFIG_IP_ALWAYS_DEFRAG option in the Linux kernel), some networks can't afford the performance hit this causes and thus leave it disabled. Note that I do not yet have this option working on all systems. It works fine for my Linux, FreeBSD, and OpenBSD boxes and some people have reported success with other *NIX variants.

-v Verbose mode. This is a highly recommended option and it gives out more information about what is going on. You can use it twice for greater effect. Use **-d** a couple of times if you really want to get crazy with scrolling the screen!

-h This handy option display a quick reference screen of nmap usage options. As you may have noticed, this man page is not exactly a 'quick reference'.

-oN <logfilename> This logs the results of your scans in a normal human readable form into the file you specify as an argument.

-oM <logfilename> This logs the results of your scans in a **machine parseable** form into the file you specify as an argument. You can give the argument '-' (without quotes) to shoot output into stdout (for shell pipelines, etc). In this case normal output will be suppressed. Watch out for error messages if you use this (they will still go to stderr). Also note that '-v' will cause some extra information to be printed.

--resume <logfilename> A network scan that is cancelled due to control-C, network outage, etc. can be resumed using this option. The logfilename must be either a normal (-oN) or machine parsable (-oM) log from the aborted scan. No other options can be given (they will be the same as the aborted scan). Nmap will start on the machine after the last one successfully scanned in the log file.

-iL <inputfilename> Reads target specifications from the file specified RATHER than from the command line. The file should contain a list of host or network expressions separated by spaces, tabs, or newlines. Use a hyphen (-) as *inputfilename* if you want nmap to read host expressions from stdin (like at the end

of a pipe). See the section *target specification* for more information on the expressions you fill the file with.

-iR This option tells Nmap to generate its own hosts to scan by simply picking random numbers. It will never end. This can be useful for statistical sampling of the Internet to estimate various things. If you are ever really bored, try *nmap -sS -iR -p 80* to find some web servers to look at.

-p <port ranges> This option specifies what ports you want to specify. For example '-p 23' will only try port 23 of the target host(s). '-p 20-30,139,60000-' scans ports between 20 and 30, port 139, and all ports greater than 60000. The default is to scan all ports between 1 and 1024 as well as any ports listed in the services file which comes with nmap.

-F Fast scan mode.

Specifies that you only wish to scan for ports listed in the services file which comes with nmap. This is obviously much faster than scanning all 65535 ports on a host.

-D <decoy1 [,decoy2][,ME],...> Causes a decoy scan to be performed which makes it appear to the remote host that the host(s) you specify as decoys are scanning the target network too. Thus their IDS might report 5-10 port scans from unique IP addresses, but they won't know which IP was scanning them and which were innocent decoys. While this can be defeated through router path tracing, response-dropping, and other "active" mechanisms, it is generally an extremely effective

technique for hiding your IP address. Separate each decoy host with commas, and you can optionally use 'ME' as one of the decoys to represent the position you want your IP address to be used. If your put 'ME' in the 6th position or later, some common port scan detectors (such as Solar Designer's excellent scanlogd) are unlikely to show your IP address at all. If you don't use 'ME', nmap will put you in a random position. Note that the hosts you use as decoys should be up or you might accidentally SYN flood your targets. Also it will be pretty easy to determine which host is scanning if only one is actually up on the network. You might want to use IP addresses instead of names (so the decoy networks don't see you in their nameserver logs). Also note that some (stupid) "port scan detectors" will firewall/deny routing to hosts that attempt port scans. Thus you might inadvertently cause the machine you scan to lose connectivity with the decoy machines you are using. This could cause the target machines major problems if the decoy is, say, its internet gateway or even "localhost". Thus you might want to be careful of this option. The real moral of the story is that detectors of spoofable port scans should not take action against the machine that seems like it is port scanning them. It could just be a decoy! Decoys are used both in the initial ping scan (using ICMP, SYN, ACK, or whatever) and during the actual port scanning phase. Decoys are also used during remote OS detection (**-O**). It is worth noting that using too many decoys may slow your scan and potentially even make it less accurate. Also, some ISPs will filter out your spoofed packets, although many (currently most) do not restrict spoofed IP packets at all.

-S <IP_Address> In some circumstances, *nmap* may not be able to determine your source address (*nmap* will tell you if this is the case). In this situation, use -S with your IP address (of the interface you wish to send packets through). Another possible use of this flag is to spoof the scan to make the targets think that **someone else** is scanning them. Imagine a company being repeatedly port scanned by a competitor! This is not a supported usage (or the main purpose) of this flag. I just think it raises an interesting possibility that people should be aware of before they go accusing others of port scanning them. **-e** would generally be required for this sort of usage.

-e <interface> Tells nmap what interface to send and receive packets on. Nmap should be able to detect this but it will tell you if it cannot.

-g <portnumber> Sets the source port number used in scans. Many naive firewall and packet filter installations make an exception in their rule set to allow DNS (53) or FTP-DATA (20) packets to come through and establish a connection. Obviously this completely subverts the security advantages of the firewall since intruders can just masquerade as FTP or DNS by modifying their source port. Obviously for a UDP scan you should try 53 first and TCP scans should try 20 before 53. Note that this is only a request nmap will honor it only if and when it is able to. For example, you can't do TCP ISN sampling all from one host: port to one host: port, so nmap changes the source port even if you used -g. Be aware that there is a small performance penalty on some scans for using this option, because I sometimes store useful information in the source port number.

-r Tells Nmap **NOT** to randomize the order in which ports are scanned.

--randomize_hosts Tells Nmap to shuffle each group of up to 2048 hosts before it scans them. This can make the scans less obvious to various network monitoring systems, especially when you combine it with slow timing options (see below).

-M <max sockets> Sets the maximum number of sockets that will be used in parallel for a TCP connect() scan (the default). This is useful to slow down the scan a little bit and avoid crashing remote machines. Another approach is to use -sS, which is generally easier for machines to handle.

TIMING OPTIONS

Generally Nmap does a good job at adjusting for Network characteristics at runtime and scanning as fast as possible while minimizing that chances of hosts/ports going undetected. However, there are same cases where Nmap's default timing policy may not meet your objectives. The following options provide a fine level of control over the scan timing:

-T <Paranoid|Sneaky|Polite|Normal|Aggressive|Insane> These are canned timing policies for conveniently expressing your priorities to Nmap. **Paranoid** mode scans **very** slowly in the hopes of avoiding detection by IDS systems. It serializes all scans (no parallel scanning) and generally waits at least 5 minutes between sending packets. **Sneaky** is similar, except it only waits 15 seconds between sending packets. **Polite** is meant to ease load on the network and reduce the chances of crashing machines. It serializes the probes and waits **at least** 0.4 seconds between them.

Normal is the default Nmap behavior, which tries to run as quickly as possible without overloading the network or missing hosts/ports. **Aggressive** mode adds a 5 minute timeout per host and it never waits more than 1.25 seconds for probe responses. **Insane** is only suitable for very fast networks or where you don't mind losing some information. It times out hosts in 75 seconds and only waits 0.3 seconds for individual probes. It does allow for very quick network sweeps though :). You can also reference these by number (0-5). For example, '-T 0' gives you Paranoid mode and '-T 5' is Insane mode. These canned timing modes should NOT be used in combination with the lower level controls given below.

--host_timeout <milliseconds> Specifies the amount of time Nmap is allowed to spend scanning a single host before giving up on that IP. The default timing mode has no host timeout.

--max_rtt_timeout <milliseconds> Specifies the maximum amount of time Nmap is allowed to wait for a probe response before retransmitting or timing out that particular probe. The default mode sets this to about 9000.

--min_rtt_timeout <milliseconds> When the target hosts start to establish a pattern of responding very quickly, Nmap will shrink the amount of time given per probe. This speeds up the scan, but can lead to missed packets when a response takes longer than usual. With this parameter you can guarantee that Nmap will wait at least the given amount of time before giving up on a probe.

--initial_rtt_timeout <milliseconds> Specifies the initial probe timeout. This is generally only useful when scanning firewalled hosts with -P0. Normally Nmap can obtain good RTT estimates from the ping and the first few probes. The default mode uses 6000.

--max_parallelism <number> Specifies the maximum number of scans Nmap is allowed to perform in parallel. Setting this to one means Nmap will never try to scan more than one port at a time. It also effects other parallel scans such as ping sweep, RPC scan, etc.

--scan_delay <milliseconds> Specifies the **minimum** amount of time Nmap must wait between probes. This is mostly useful to reduce network load or to slow the scan way down to sneak under IDS thresholds. (Granquist, 1999)

APPENDIX C: GLOSSARY

Abuse of Privilege:

When a user performs an action that they should not have, according to organizational policy or law.

Access Control Lists:

Rules for packet filters (typically routers) that define which packets to pass and which to block.

Access Router:

A router that connects your network to the external Internet. Typically, this is your first line of defense against attackers from the outside Internet. By enabling access control lists on this router, you'll be able to provide a level of protection for all of the hosts "behind" that router, effectively making that network a DMZ instead of an unprotected external LAN.

Application-Level Firewall:

A firewall system in which service is provided by processes that maintain complete TCP connection state and sequencing. Application level firewalls often re-address traffic so that outgoing traffic appears to have originated from the firewall, rather than the internal host.

Authentication:

The process of determining the identity of a user that is attempting to access a system.

Authentication Token:

A portable device used for authenticating a user. Authentication tokens operate by challenge/response, time-based code sequences, or other techniques. This may include paper-based lists of one-time passwords.

Authorization:

The process of determining what types of activities are permitted. Usually, authorization is in the context of authentication: once you have authenticated a user, they may be authorized different types of access or activity.

Bastion Host:

A system that has been hardened to resist attack, and which is installed on a network in such a way that it is expected to potentially come under attack. Bastion hosts are often components of firewalls, or may be "outside" Web servers or public access systems. Generally, a bastion host is running some form of general purpose operating system (e.g., Unix, VMS, NT, etc.) rather than a ROM-based or firmware operating system.

Challenge/Response:

An authentication technique whereby a server sends an unpredictable challenge to the user, who computes a response using some form of authentication token.

Chroot:

A technique under Unix whereby a process is permanently restricted to an isolated subset of the file system.

Cryptographic Checksum:

A one-way function applied to a file to produce a unique "fingerprint" of the file for later reference. Checksum systems are a primary means of detecting file system tampering on Unix.

Data Driven Attack:

A form of attack in which the attack is encoded in innocuous-seeming data which is executed by a user or other software to implement an attack. In the case of firewalls, a data driven attack is a concern since it may get through the firewall in data form and launch an attack against a system behind the firewall.

Defense in Depth:

The security approach whereby each system on the network is secured to the greatest possible degree. May be used in conjunction with firewalls.

DNS spoofing:

Assuming the DNS name of another system by either corrupting the name service cache of a victim system, or by compromising a domain name server for a valid domain.

Dual Homed Gateway:

A dual homed gateway is a system that has two or more network interfaces, each of which is connected to a different network. In firewall configurations, a dual homed gateway usually acts to block or filter some or all of the traffic trying to pass between the networks.

Encrypting Router:

see Tunneling Router and Virtual Network Perimeter.

Firewall:

A system or combination of systems that enforces a boundary between two or more networks.

Host-based Security:

The technique of securing an individual system from attack. Host based security is operating system and version dependent.

Insider Attack:

An attack originating from inside a protected network.

Intrusion Detection:

Detection of break-ins or break-in attempts either manually or via software expert systems that operate on logs or other information available on the network.

IP Spoofing:

An attack whereby a system attempts to illicitly impersonate another system by using its IP network address.

IP Splicing / Hijacking:

An attack whereby an active, established, session is intercepted and co-opted by the attacker. IP Splicing attacks may occur after an authentication has been made, permitting the attacker to assume the role of an already authorized user. Primary protections against IP Splicing rely on encryption at the session or network layer.

Least Privilege:

Designing operational aspects of a system to operate with a minimum amount of system privilege. This reduces the authorization level at which various actions are performed and decreases the chance that a process or user with high privileges may be caused to perform unauthorized activity resulting in a security breach.

Logging:

The process of storing information about events that occurred on the firewall or network.

Log Retention:

How long audit logs are retained and maintained.

Log Processing:

How audit logs are processed, searched for key events, or summarized.

Network-Level Firewall:

A firewall in which traffic is examined at the network protocol packet level.

Perimeter-based Security:

The technique of securing a network by controlling access to all entry and exit points of the network.

Policy:

Organization-level rules governing acceptable use of computing resources, security practices, and operational procedures.

Proxy:

A software agent that acts on behalf of a user. Typical proxies accept a connection from a user, make a decision as to whether or not the user or client IP address is permitted to use the proxy, perhaps does additional authentication, and then completes a connection on behalf of the user to a remote destination.

Screened Host:

A host on a network behind a screening router. The degree to which a screened host may be accessed depends on the screening rules in the router.

Screened Subnet:

A subnet behind a screening router. The degree to which the subnet may be accessed depends on the screening rules in the router.

Screening Router:

A router configured to permit or deny traffic based on a set of permission rules installed by the administrator.

Session Stealing:

See IP Splicing.

Trojan Horse:

A software entity that appears to do something normal but which, in fact, contains a trapdoor or attack program.

Tunneling Router:

A router or system capable of routing traffic by encrypting it and encapsulating it for transmission across an untrusted network, for eventual de-encapsulation and decryption.

Social Engineering:

An attack based on deceiving users or administrators at the target site. Social engineering attacks are typically carried out by telephoning users or operators and pretending to be an authorized user, to attempt to gain illicit access to systems.

Virtual Network Perimeter:

A network that appears to be a single protected network behind firewalls, which actually encompasses encrypted virtual links over untrusted networks.

Virus:

A replicating code segment that attaches itself to a program or data file. Viruses might or might not contain attack programs or trapdoors. Unfortunately, many have taken to calling *any* malicious code a "virus". If you mean "Trojan horse" or "worm", say "Trojan horse" or "worm".

Worm:

A stand alone program that, when run, copies itself from one host to another, and then runs itself on each newly infected host. The widely reported "Internet Virus" of 1988 was not a virus at all, but actually a worm.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Anonymous, Maximum Linux Security, pp. 487-509, Sams Publishing, Indianapolis, IN, 2000

2. Barkakati, Naba, Red Hat Linux Secrets, pp. 569, 621, 623, IDG Books Worldwide, Foster City, CA, 1999

3. Brenton, C., Mastering Network Security, Network Press, San Francisco, California, 1999

4. Breyer, Robert and Riley, Sean, Switched and Fast Ethernet: How It Works and How to Use It, pp. 174-211, Ziff-Davis Press, Emeryville, CA, 1995

5. Cheswick, William R. and Bellovin, Steven M., Firewalls and Internet Security and Internet Security: Repelling the Wiley Hacker, pp. 271-277, Addison~Wesley Publishing, Menlo Park, CA, 1994

6. Creed, Adam, LinuxOne Expands into Taiwan, Daily News, p. 1, January 6, 2000

7. Danesh, Arman, Red Hat Linux 6, pp. 179-187, 656-658,Sybex, San Francisco, CA, 1999

8. Edwards, Mark Joseph, Internet Security With Windows NT, pp. 25-33, Duke Communications Worldwide, Loveland, CO, 1998

9. Escamilla, Terry, Intrusion Detection , pp. 131-135, Wiley Computer Publishing, New York, NY, USA, 1998

10. Goncalves, Marcus, Protecting Your Web Site With Firewalls, pp. 81-84, Prentice Hall, Upper Saddle River, NJ, USA, 1997

11. Granquist, Lamont, Nmap Tutorial Guide, http//www.insecure.com/nmap/lamont-nmap-guide.txt, 5 April 99

12. Hallberg, Bruce, Networking A Beginner's Guide, pp. 100-112, Osborne/McGraw Hill, Berkeley, CA, 2000

13. Honegger, Barbara, NPS Named Center of Academic Excellence for INFOSEC, Campus News, Vol. 6 Issue 19, p. 1, May 11, 2000

14. Kyas, Othmar, Internet Security, pp. 18-22, International Thompson Publishing, Boston, MA, USA, 1997

15. Levy, Steven & Stone, Brad, Hunting the Hackers, Newsweek, pp. 38-44, February 21, 2000

16. Lowe, Doug, Networking for Dummies, pp. 213-214, 300, IDG Books Worldwide, Foster City, CA, 1999

17. Loshin, Pete, Securing Linux, Information Security, pp. 20-31, Feb 2000

18. Maney, Kevin & McMahon, Patrick, 'Love Bug' Virus Created in Ordinary Petri Dish, USA Today, p. 2, May 15, 2000

19. McClure, Stuart & Scambray, Joel, Kurtz, George, Hacking Exposed Network Security Secrets & Solutions, pp. 4-85, Osborne/McGraw Hill, Berkeley, CA, 1999

20. Schwoerer, Sean, Linux as a Firewall Solution?, Firewalls 101: Perimeter Protection With Firewalls, SANS Institute, Network Security Conference, San Jose, CA, pp. 4-1-4-55, May 9, 2000

21. Shah, Steve, Linux Administration A Beginner's Guide, pp. 476-483, Osborne/McGraw Hill, Berkeley, CA, 2000

22. Siyan, Karanjit & Ware, Chris, Internet Firewalls and Network Security, pp. 91-96, New Riders Publishing, Indianapolis, IN, 1995

23. Sobell, Mark G., A Practical Guide to Linux, pp. 3-17, Addison-Wesley, Berkeley, CA, 1997

24. Staff, Compaq to Develop Chinese Linux, IT Daily, [http://computercurrents.com/newstoday/99/09/01/news4.html]. Sept 1, 1999

25. Staff, Red Hat Expands into Japan, Daily News, [http://www.currents.net/news/99/09/09/news9.html]. Sept 9, 1999

26. Staff Reporter, Linux to Pay Off, Daily News, [http://www.currents.net/news/00/03/10/news9.html]. March 10, 2000

27. Stallings, William, Data and Computer Communications (Fifth Edition), pp. 211-216, Prentice Hall, Upper Saddle River, NJ, 1997

28. Stallings William, Internet Security Handbook, pp. 45-49, IDG Books Worldwide, Foster City, CA, 1995
29. Witherspoon, Coletta and Craig, Red Hat Linux 6: fast & easy, pp. 170-173, 142-151, Prima Tech, Rocklin, CA, 1999

30. Ziegler, Robert, Linux Firewalls, pp. 62-126, New Riders Publishing, Indianapolis, IN, 2000

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library.....2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Raymond Bernstein, Code/ECBe.....1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

4. Vicente Garcia, Code/ECGa.....1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

5. Chairman, Code EC.....1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

6. Bryan S. Lopez.....2
   217 S. Tornillo St.
   Las Cruces, NM 88001

66 290NPG TH 2745
6/02 22527-200 NLB

DUDLEY KNOX LIBRARY
3 2768 00403849 7